Universitat de Vic
Escola Politècnica Superior

Treball Final de Carrera

Control de respiració en temps real per dispositius Android

Amador Criado Rozas

Enginyeria Tècnica d'Informàtica de Sistemes
Director: Gerard Maferrer
Vic, juny de 2014

Títol:	Control de respiració en temps real per dispositius Android
Autor:	Amador Criado Rozas
Director:	Gerard Masferrer
Data:	02-juny-2014

Resum:

El projecte està basat en la creació d'una aplicació per dispositius mòbils android i que fent servir l'ús del micròfon capturi el so que genera l'usuari i pugui determinar si s'està respirant i en quin punt de la respiració es troba l'usuari.

S'ha dut a terme una filosofia de disseny orientada a l'usuari (DCU) de manera que el primer pas ha sigut realitzar un prototip i un 'sketch'. A continuació, s'han realitzat 10 aplicacions test i en cadascuna d'elles s'ha ampliat la funcionalitat fins a arribar a obtenir una aplicació base que s'aproxima al disseny inicial generat per mitjà del prototip.

El més important dels dissenys algorísmics que s'han realitzat per la aplicació es la capacitat de processar el senyal en temps real, ja que fins i tot s'ha pogut aplicar la transformada ràpida de Fourier (FFT) en temps real sense que el rendiment de l'aplicació es veies afectat.

Això ha sigut possible gràcies al disseny del processament amb doble buffer i amb un fil d'execució dedicat independent del fil principal d'execució del programa 'UI Thread'

Títle:	Control of breathing in real time to Android devices
Author:	Amador Criado Rozas
Director:	Gerard Masferrer
Dat3:	02-june-2014

Summary:

The project is based on creating an application for mobile devices using Android and the microphone. Capturing the sound generated by the user can determine if you are breathing in and how breathing is the user .

Has carried out a philosophy of user-oriented design (UCD) so the first step was to make a prototype and a sketch. Then , there have been 10 applications and test each functionality has been extended up to get a basic application that approaches the initial design generated by the prototype.

The most important algorithmic designs that have been made to the application 's ability to process the signal in real time, even as it failed to apply the Fast Fourier Transform (FFT) in real time without performance application would be affected .

This has been possible thanks to the design of processing double buffer and a separate dedicated thread of execution of the main thread of execution of the program UI Thread.

Índex de continguts

1. Introducció

La constant innovació tecnològica que la societat està vivint de forma permanent ha derivat en l'aparició de nous dispositius que cada cop més, estan facilitant la vida dels seus usuaris i que esta aportant eines per poder augmentar la productivitat i disminuir qualsevol interval de no aprofitament del temps durant el dia... Des de smartphone, tablet PC, ordinadors portàtils o netbooks, el ventall de dispositius es cada vegada més ampli i a tot això s'ha d'afegir l'aparició i implantació definitiva en els últims anys de la xarxa sense fils 3G que cada cop disposa de més cobertura, fet que permet una comunicació molt potent i un accés a internet des d'un percentatge molt alt de la geografia mundial.

En tota aquesta evolució, es va plantejar el propòsit principal del projecte posant el focus en els dispositius mòbils intel·ligents o smartphone, que poc a poc estan substituint els ordinadors de sobretaula en tasques tan importants com l'accés a la xarxa. A continuació es detallaran les principals causes per les quals s'ha escollit dur a terme el projecte en un dispositiu android, i veurem la importància que amb el pas del temps, estan adquirint a nivell mundial:

1.1 Perquè s'ha escollit un smartphone?

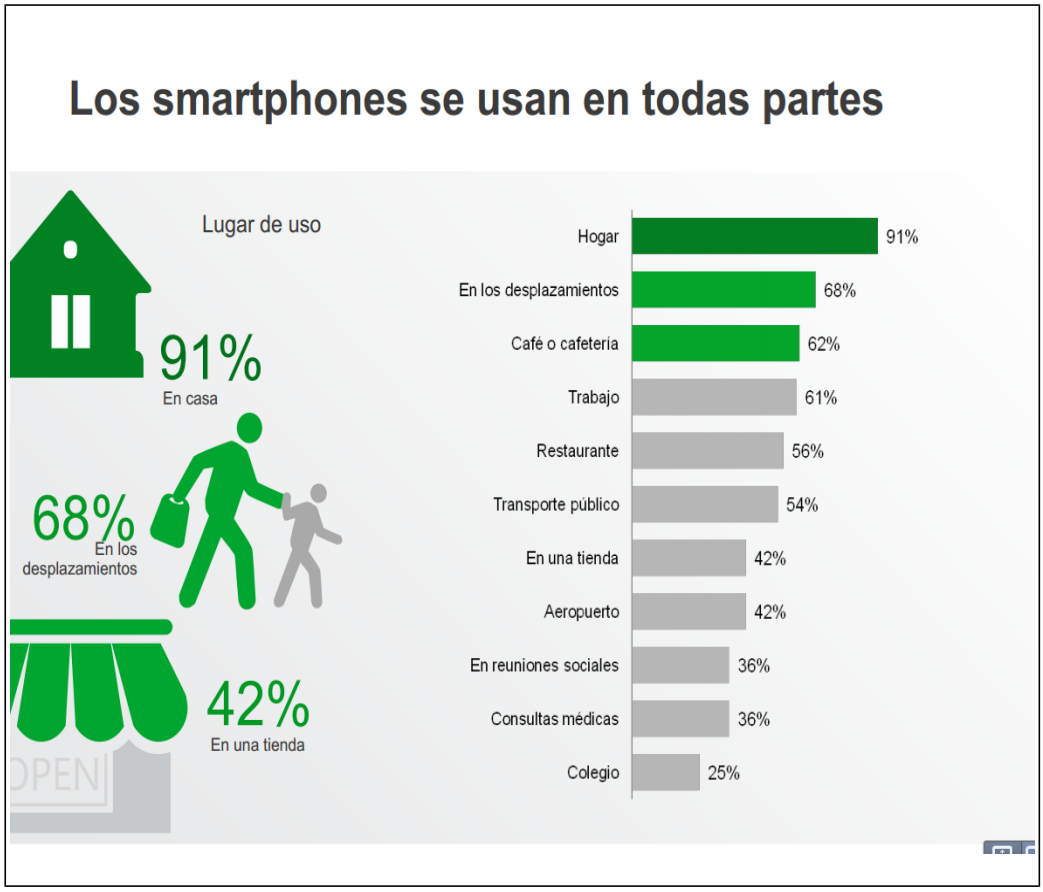
1. S'han convertit en una **eina imprescindibl**e a la nostra vida quotidiana, la societat s'expressa mitjançat aquests dispositius i s'està produint un creixement personal i tecnològic dels usuaris gràcies a poder disposar d'un ordinador de butxaca connectat a internet en qualsevol indret... a tot això, podríem destacar que aquests ordinadors de butxaca, poden fins i tot fer trucades!

2. Per les empreses **s'ha convertit en un mitjà importantíssim per arribar al client**. Està clar que totes les empreses que apostin per adaptar la seva estratègia a les comunicacions mòbils i estiguin a l'avantguarda en aplicacions mòbils es beneficaran de la situació actual.

3. **Han transformat en gran part el comportament del consumidor**, cerca mitjançant el mòbil, visualització de vídeos i les xarxes socials estan augmentant cada vegada amés i ocupen molt més el temps dels usuaris que abans.

4. **Els smartphone han canviat la forma en que els usuaris compren**, Les cerques de productes mitjançant el mòbil estan augmentant considerablement, el cercador de Google rep més peticions de cerca des d'aquests dispositius que des de ordinadors de sobretaula

5. **Es fan servir en qualsevol indret**, Això ha permès que els usuaris puguin moure's amb més independència de la tecnologia ja que aquesta l'acompanya. A més, beneficiar-se d'amplificatius GPS o aplicacions orientades a la ubicació local han suposat una revolució .



(Abril 2014) Enquesta de Our Mobile plane: España "Conoce mejor al consumidor mòbil"

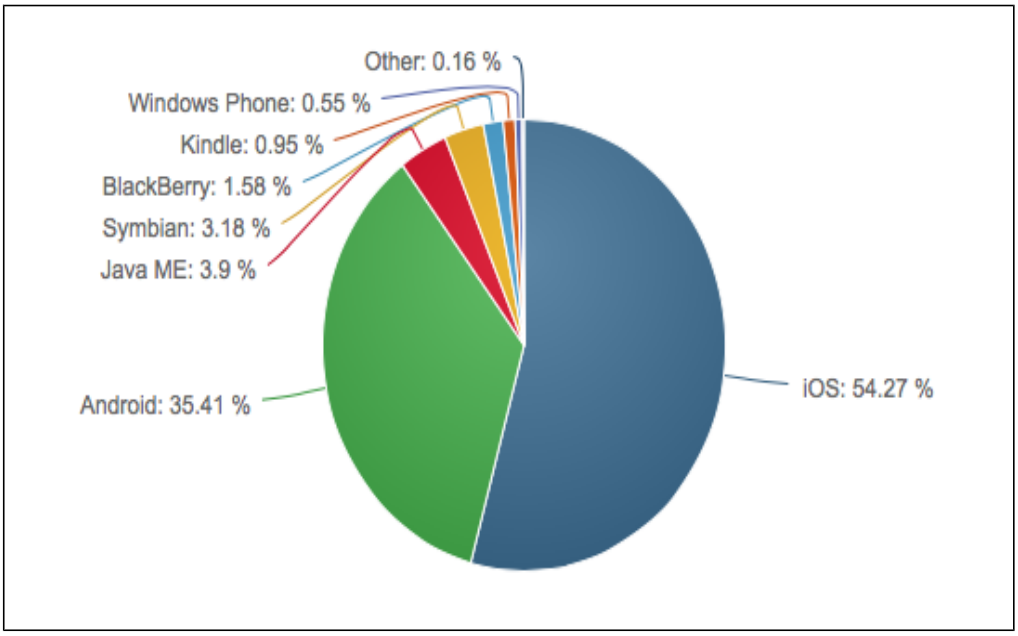
1.2 Perquè Android?

Android és un sistema operatiu mòbil basat en Linux enfocat per ser utilitzat en dispositius mòbils com telèfons intel·ligents, tauletes, Google TV i altres dispositius . És desenvolupat per la *Open Handset Alliance* , liderada per *Google* .
L'estructura del sistema operatiu Android es compon d'aplicacions que s'executen en un *framework* Java d'aplicacions orientades a objectes sobre el nucli de les biblioteques de *Java* en una màquina virtual *Dalvik* amb compilació en temps d'execució . Les biblioteques escrites en llenguatge C inclouen un administrador d'interfície gràfica, un *framework OpenCore* , un base de dades relacional *SQLite* , una Interfície de programació *d'API* gràfica *OpenGL* ÉS 2.0 3D , un motor de renderitzat *WebKit* , un motor gràfic *SGL* , *SSL* i una biblioteca estàndard de C *Bionic* .

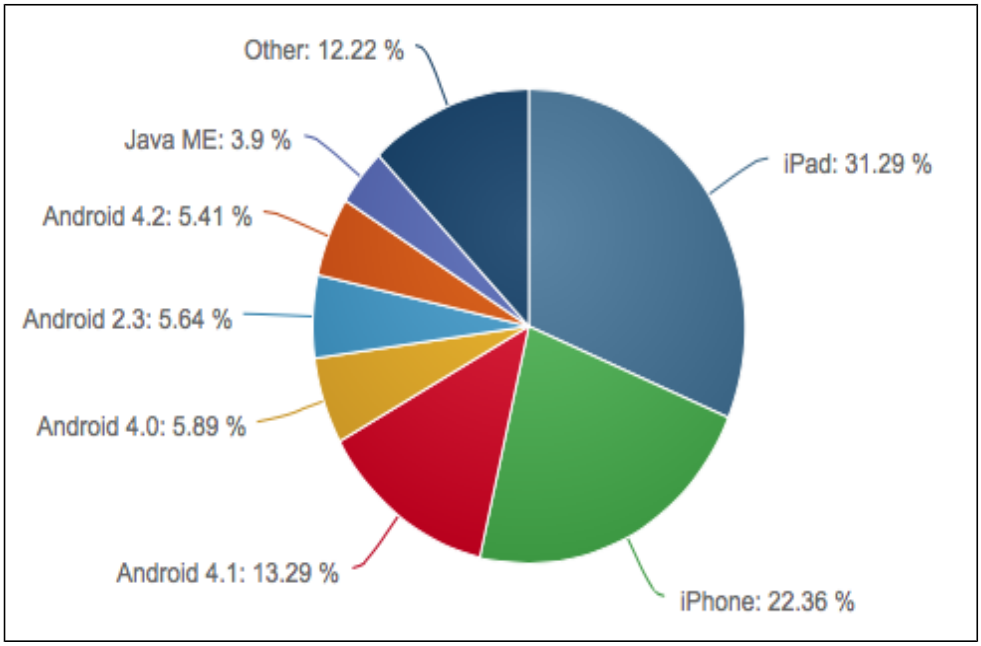
Les aplicacions es desenvolupen habitualment en el llenguatge *Java* amb *Android* Software *Development Kit* (Android SDK) , Hi ha altres eines de desenvolupament , incloent un Kit de Desenvolupament Nadiu per aplicacions o extensions en *C* , *C++* o altres llenguatges de programació .
Android desenvolupa de forma oberta i es pot accedir tant al codi font com a la llista d'incidències on es poden veure problemes encara no resolts i reportar problemes nous . En l'actualitat existeixen més de 700.000 aplicacions per Android i s'estima que 1.000.000 telèfons mòbils s'activen diàriament . Android és criticat per la fragmentació que pateixen

els seus terminals en no ser suportats amb actualitzacions per part dels fabricants , cosa que es podria considerar obsolescència programada . Això suposa problemes per als desenvolupadors que han de bregar amb la retro-compatibilitat entre diferents versions del sistema operatiu . Resulta evident que aprendre a desenvolupar projectes per a aquest sistema operatiu millora les perspectives tant laborals com acadèmiques de qualsevol estudiant d'Informàtica o similars . Per desenvolupar sobre Android es farà ús del paquet *ADT* (Android Developer Tools), en la seva versió *Bundle*, que inclou el programari necessari per començar desenvolupar per a aquesta plataforma .

Situació actual dels sistemes operatius per dispositius mòbils:



(gener 2014) Dades recollides de NetMarketShare



(gener 2014) Dades recollides de NetMarketShare

1.3 La idea i Objectius

Podríem dir que els smartphone són el medi de transport i que Android és el motor per arribar a assolir qualsevol idea i convertir-la en una utilitat universal per qualsevol usuari d'android. La traducció menys abstracta de tot això seria el que coneixem com a una aplicació.
En el cas d'aquest projecte la idea que ha derivat en aplicació ha sigut poder assolir una funcionalitat poc coneguda però no per això de menys importància que d'altres. Es tracta de poder controlar la respiració de l'usuari mitjançant el dispositiu mòbil, i això es durà a terme desenvolupant una aplicació amb aquesta finalitat.
Són molts els beneficis de poder controlar la respiració, el fet de que sigui instintiu en l'ésser humà no implica que en determinats àmbits i amb alguns propòsits, sigui molt important aprendre a controlar o pautar la respiració d'un individu. A continuació detallarem algunes aplicacions importants:

1- Tècniques de concentració.

2- Controlar la Ira.

3- Tècniques de relaxació (respiració diafragmàtica).

4- Millorar la respiració.

5- Respirar millora la salud (nens amb fibrosis).

6- Fisioteràpia respiratòria.

***Objectius*:**
A continuació es detallaran els objectius principals del projecte. S'ha destacar que s'han escollit continguts que puguin ser aplicats des dels coneixements adquirits durant l'estudi de la enginyeria, però també s'ha intentat donar un pas més enllà i poder aprendre nous coneixements, ja que molts dels objectius que s'han assolit formen part de sistemes de molta actualitat i ha calgut una investigació i aprenentatge abans d'assolir els objectius.
Es per això que s'ha orientat el projecte en dos parts ben diferenciades, que serien, per una banda el 'testing' i per l'altre la creació d'aplicacions definitives a partir de la investigació i estudi realitzat al 'testing' previ. També s'han afegit alguns annexes sobre possibilitats de millora de l'aplicació, sobretot en termes de disseny. De totes formes detallarem la metodologia que s'ha fet servir més endavant.

Aquest són els principals objectius del projecte:

- Es vol aconseguir una aplicació capaç de captar la respiració de l'usuari en temps real.
- Aprendre a programar aplicacions Android sobre l'entorn de desenvolupament Eclipse i en llenguatge Java.
- Es farà servir doble buffer i processament en temps d'execució.
- Aplicació de la transformada de Fourier FFT en temps real.
- Desenvolupar aplicacions amb una interfície senzilla i intuïtiva per l'usuari.
- Aconseguir una llibreria amb les principals classes utilitzades al projecte.
- Disseny d'aplicacions compatible amb smartphone i Tablet PC.

2. Disseny i Metodologia

El projecte ha constat de diferents etapes, a continuació les enumerarem però cal destacar que tot i que les hem detallat, podríem agrupar la metodologia del projecte en tres parts ben diferenciades:

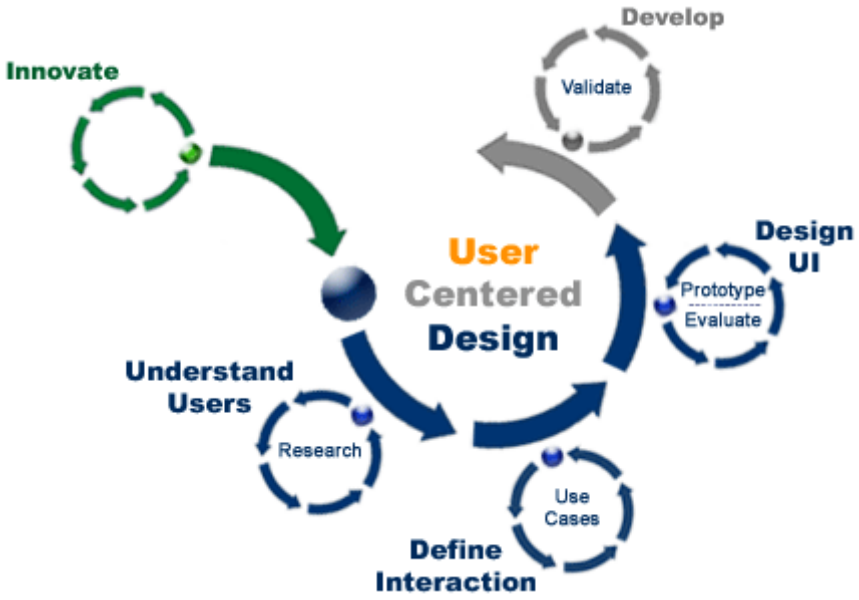
1- Disseny inicial i Prototipat. (Inclou l'anàlisi).
2- Disseny Algorísmic i Implementació.
3- Conclusions

Metodologia detallada:

- S'ha escollit la filosofia de disseny DCU (Disseny centrat en l'usuari)
- Disseny de prototip : Aplicació per smartphone i Tablet que enregistri i monitoritzi la respiració en temps real.
- Anàlisi de requisits funcionals i no funcionals del prototip dissenyat.
- 'Testing' entre diversos usuaris.
- El llenguatge de programació és Java i el SDK d'Android per dispositius mòbils
- Disseny algorítmic d'un sistema de doble buffer que mantingui el processament en temps real de l'aplicació.
- Implementació d'aplicacions TEST amb funcionalitat progressiva.
- S'ha triat Eclipse com a entorn de desenvolupament (IDE) i el sistema Operatiu on s'ha realitzat el desenvolupament es Ubuntu 12.04 64 bits
- les implementacions de les aplicacions s'han depurat en els dispositius: Samsung Galaxy Note II / XCSOURCE 10.1 / HTC Desire
- Es crearan llibreries amb les classes més importants del projecte. A més S'ha generat una documentació JavaDoc corresponent.
- Annexes i possibilitats de millora

2.1 Filosofia de Disseny (DCU)

Abans de començar qualsevol projecte de software és important determinar quina filosofia de disseny es durà a terme, en aquest cas s'ha escollit una filosofia de disseny DCU (*Disseny Centrat en l'Usuari*), ja que es pretén que les aplicacions resultants puguin resoldre necessitats concretes dels propis usuaris, aconseguint d'aquesta forma una major satisfacció i una experiència d'us amb el mínim esforç per part d'aquests.



La majoria dels processos que fan Disseny Centrat en l'Usuari suposen les següents etapes:

1. Conèixer a fons als usuaris finals, haurem de definir un 'target' i analitzar si s'ajusta bé a el model d'aplicació que es vol desenvolupar.

2. Dissenyar un producte que resolgui les seves necessitats i s'ajusti a les seves capacitats, expectatives i motivacions

3. Posar a prova el dissenyat, normalment usant test d'usuaris

El motiu de la elecció d'aquest tipus de Metodologia es evident, s'ha destacat la importància de les aplicacions mòbils en la societat actual, com ha augmentat la importància dels dispositius en el dia a dia dels usuaris, per tant, no podem oblidar-nos de que l'experiència d'usuari i la usabilitat son factors molt importants.

La clau del disseny DCU, és que s'ha de tenir en compte l'usuari en totes les fases de creació dels Aplicatius, per començar en el procés previ de 'brainstorming' ja s'ha tingut en compte l'usuari considerablement.

S'han sondejat diversos usuaris tester de que disposen de perfils específics, s'ha intentat que no siguin del mateix àmbit tot i que s'ha valorat que l'aplicació ha d'anar dirigida principalment a usuaris amb problemes respiratoris , que necessitin realitzar exercicis de concentració a través de la respiració o per controlar la ira o l'estres.

2.2 Disseny de prototip

La elaboració de prototips en un sistema d'informació és una tècnica valuosa per la recopilació ràpida d'informació específica que ens permeti definir els requisits que ha de tenir la nostra aplicació. S'han elaborat els prototips en una data temprana, podríem considerar l'inici del cicle de vida de l'aplicació, un cop s'ha consolidat la idea inicial i es comença a desenvolupar.

2.2.1 Que és un prototip?

- El prototip és una aplicació que funciona
- Els prototips es creen amb rapidesa
- Els prototips evolucionen a través d'un procés iteratiu
- Els prototips tenen un baix cost de desenvolupament

Per realitzar el prototipat s'ha fe servir JustinMind prototyper free, que es tracta d'una eina amb moltes possibilitats de disseny i àgil a l'hora de crear events o fluxos de l'aplicació. Ha set necessari aprendre conceptes de prototipat i també s'ha hagut d'aprendre el funcionament de l'aplicació JustinMind.

En aquest projecte s'han dissenyat prototips per dos models de dispositius mòbils, per una banda, un smartphone amb pantalla de 4” i per un altre banda, una Tablet PC també amb sistema operatiu android, amb una pantalla de 10.1”.

2.2.2 Prototip1: smartphone Android 4.4

– Smartphone Android 4.4, pantalla de 4”:

Anem a repassar els diferents estats possibles que s'han definit per aquest prototip:

0- **Loading (Carregant):**

– Pantalla inicial de càrrega de l'aplicació
– Ja podem comprovar que s'ha utilitzat un disseny minimalista i amb uns estils que volen transmetre harmonia ja que es vol brindar a l'usuari un entorn de relaxació o de concentració ideal per fer servir l'aplicació.
– S'ha escollit l'anglès com idioma base per l'aplicació per poder-la fer més accessible a un major rang d'usuaris, però en el cas de producció comercial de l'aplicació es distribuiria en Català i Espanyol també.

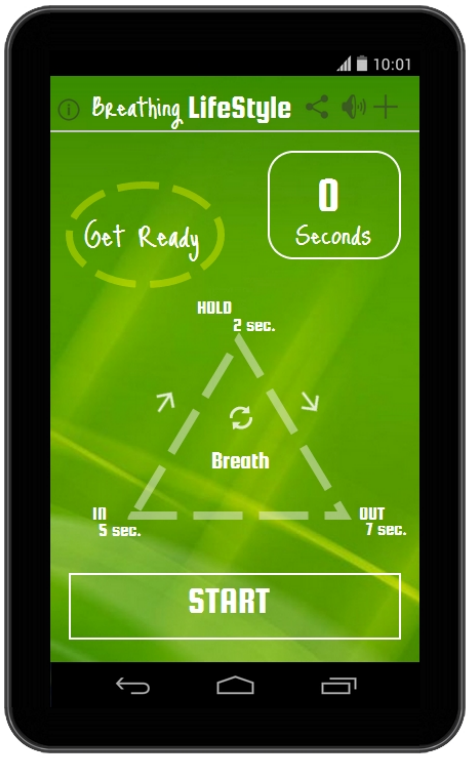
1- Splash Screen (Pantalla de Benvinguda)

- Pantalla de benvinguda, com podem comprovar, es manté el disseny minimalista i a més apareixen nous objectes com el botó de '*START*' i en la part superior de l'aplicació podem veure dos icones, una fa referència al só i l'altre es un botó per obrir les opcions del sistema.
- En aquesta pantalla sonarà de fons una musica estil '*chill-out*' per relaxar a l'usuari. Si l'usuari polsa la icona de l'altaveu es desactivaria el só.

2- DashBoard

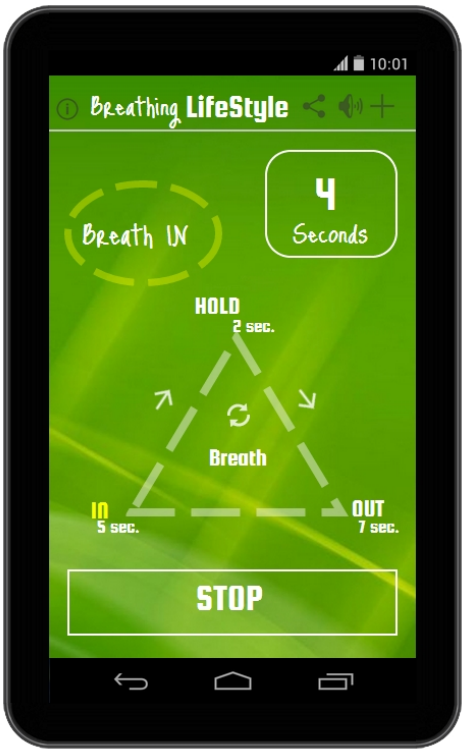
- Es la pantalla on trobem la llista d'opcions que disposa l'usuari,
- S'ha afegit un botó per compartir l'aplicació. És poden considerar diferents mètodes per compartir, com per exemple mostrar a alguna xarxa social la aplicació (*Facebook*, *Twitter*...) o enviar per *Whatsapp* o correu electrònic a un contacte l'enllaç per descarregar l'aplicació.
- S'ha afegit a cada opció, un diagrama amb les propietats de cada pauta respiratòria.
- La idea de l'estil de l'aplicació es minimalista i amb disseny lleuger i fàcil per l'usuari final.
- El codi de colors també simplifica cada secció i facilita l'ús a l'usuari.

3- Section Screen



- Aquesta pantalla serà comú a les tres opcions escollides, es el pas previ abans de començar a capturar la respiració en temps real. Es el '*screenshot*' del prototip mostrat podem observar la '*section screen*' de '*Relax*'
- S'ha de polsar '*Start*' per que l'aplicació inicii l'enregistrament d'àudio i el processament de la senyal per identificar la inspiració/expiració.

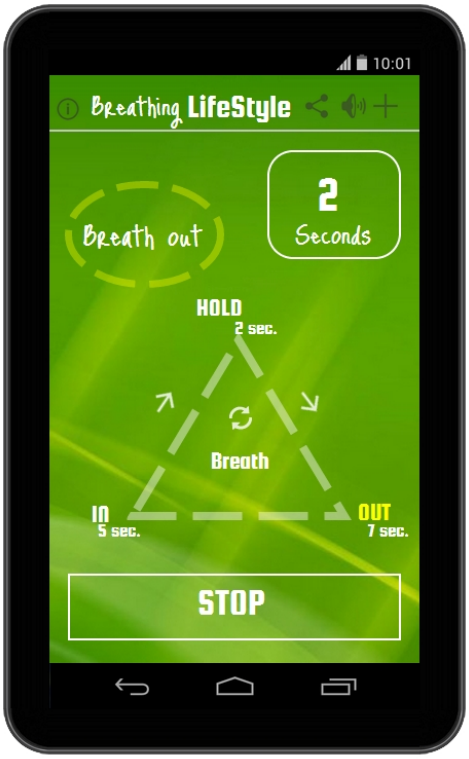
4- Breathing:



- 'Breath IN' (Inspirar)

- S'ha afegit un comptador en la part superior dreta.
- Si el sistema detecta que l'usuari no esta respirant tal com indica la pauta, i per tant, inspira o expira abans de que es compleixin els segons previstos.. L'aplicació emetrà un avís, que pot ser una vibració o tremolor de la pantalla, falta confirmar-ho mitjançant *testing* i *feed-back* dels usuaris.
- És manté el codi de colors definit en la pantalla '*splash screen*', de manera que si per exemple s'ha escollit la opció '*Relax*' els colors seran grocs, i si s'escolleix '*concentrate*' els color seran vermells per activar a l'usuari.
- Si es polsa '*STOP*' es retorna a l'usuari a la pantalla '*Section Screen*'

- **Hold (Mantenir aire)**
  - Igual que la pantalla anterior, en aquesta es tracta de mantenir l'aire inspirat anteriorment.
  - El sistema detectarà un nivell baix d'energia del só o una freqüència molt propera al silenci. En cas de no ser així, l'aplicació emetria un avís.



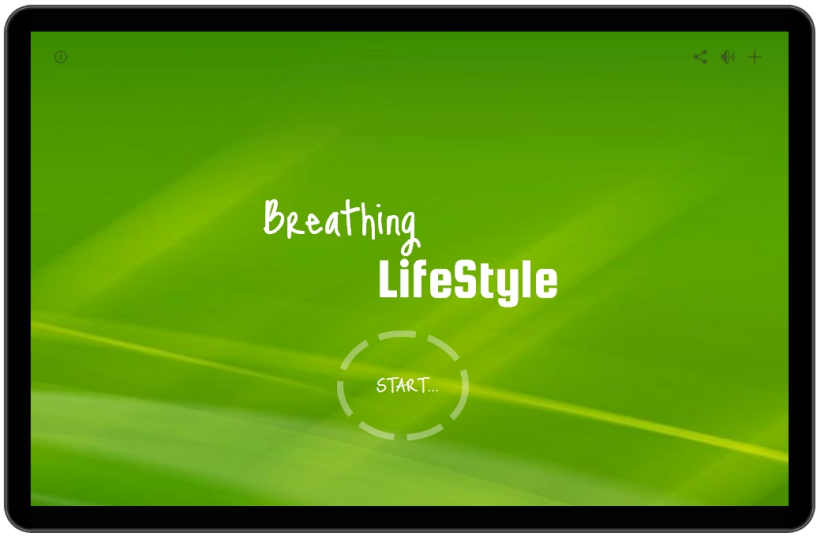
- **Breath out (Mantenir aire)**
  - S'espolsarà l'aire inspirat en l'estat '*Breath in*'
  - Té les mateixes propietats que els apartats anteriors.

2.2.3 Prototip2: Tablet PC, Android 4.1

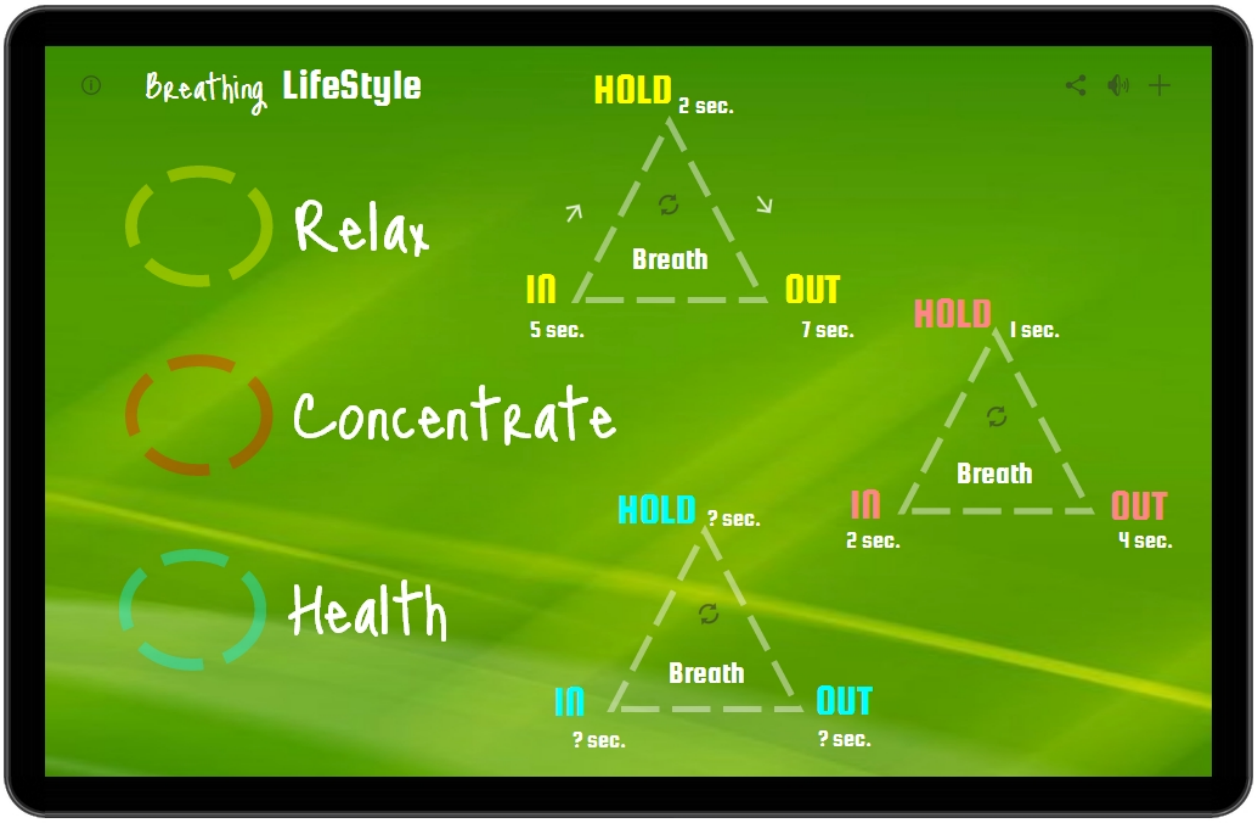
Mostrarem el disseny final del prototip adaptat a *Tablet PC*, la funcionalitat de l'aplicació es la mateixa, només s'han adaptat la mida de les icones i la distribució a una mida de pantalla més gran.

0- Loading:

1-Splash

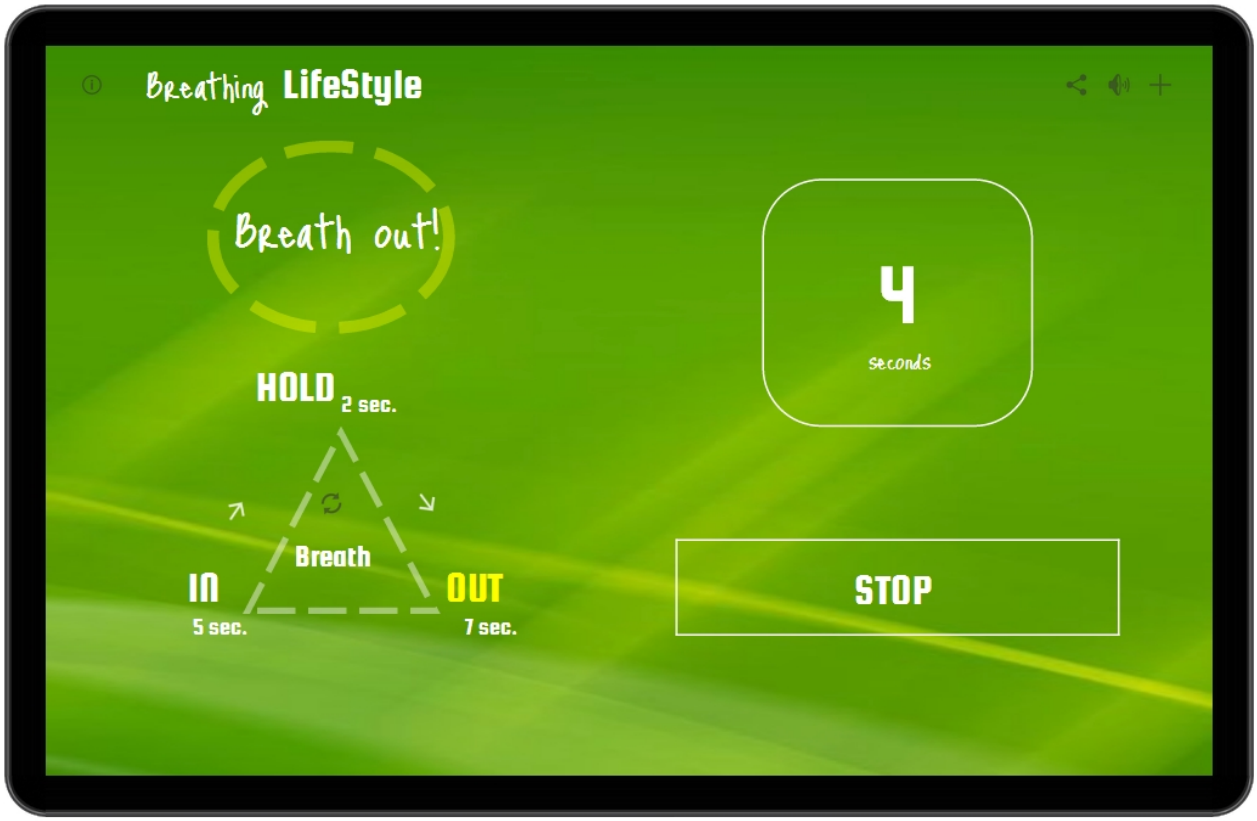


2- DashBoard



3- Section Screen

4-Breathing



2.3 Anàlisi de requeriments

Els requeriments / requisits d'un sistema descriuen els serveis que ha d'oferir el sistema i les restriccions associades al seu funcionament. Pel que fa al projecte s'han definit tant els requeriments funcionals com els no funcionals del prototip dissenyat:

2.3.1 Requeriments funcionals

Modul	funcionalitat requerida	funcionalitat objectiva
Interficie d'usuari	Seleccionar opció	mostrar a l usuari totes les possibilitats de l'aplicació en una única pantalla
	Compartir aplicació	Poder compartir l'aplicació tant a xarxes socials com a contactes de l'usuari
	Habilitar / inhabilitar só	Botó sempre visible que actua com a switch,permetent el só quan està activat i deixant la aplicació en mute quan no ho està.
	Animacions	Tant en botons, textos com imatges, s'afegiran animacions per captar millor l'atenció de l'usuari
	Info (About..)	Es mostrarà informació sobre l'aplicació i un enllaç per obrir la pàgina web de l'aplicació
	Configuració	Pantalla per parametritzar diversos aspectes de l'aplicació, per exemple, volum del só, idioma de l'aplicació...
Control de respiració	Enregistrament d'àudio	Enregistrar clips d'àudio en els buffers predefinits en el codi font de l'aplicació
	Processament d'àudio en temps real	Processament mitjançant doble buffer de els clips enregistrats en memòria durant la execució del programa
	pautes de respiració interactives	Existiran tres possibles pautes de respiració, i el flux d'execució del programa ens indicarà de forma interactiva en quin estat ens trobem. EL motiu d ela interactivitat d'aquest requeriment es que si no es segueix la pauta indicada, l'aplicació avisarà a l'usuari.
	Parametrització de pautes	En el mode 'health' es podrà establir prèviament els intervals de respiració segons les necessitats de l'usuari.

2.3.2 Requeriments no funcionals:

Concepte	Descripció objectiva
rendiment	Es imprescindible disposar d'un sistema amb capacitat de processament multi-fil., i 512 MB de RAM
disponibilitat	Possibilitat de publicar l'aplicació en el 'market' d'android(Play store). Pel que fa a les llibreries obtingudes amb les classes més utilitzades del projecte, es generara un document JavaDoc amb interfície web
seguretat	L'aplicació només necessita permisos per accedir al micròfon per enregistrar l'àudio, no cal accedir a disc per que es tracta la informació en temps real mitjançant els buffers de processament.
accessibilitat	Disseny minimalista i amb textos i botons adaptats a dispositius mòbils
usabilitat	Compatible amb qualsevol plataforma d'android a partir del 'target' 8 del SDK.
estabilitat	S'ha realitzat un procediment de 'testing' en la implementació que ha de permetre obtenir un producte final estable.
portabilitat	Gràcies a la implementació de llibreries associades a les principals classes del projecte, es por aprofitar el processament algorísmic dissenyat per implementar altres aplicacions amb funcionalitat relacionada.

3. Disseny Algorísmic i Implementació.

En aquesta etapa del projecte s'ha dut a terme la implementació de les aplicacions test i llibreries. S'ha programat de forma àgil i amb funcionalitat progressiva, augmentant el nombre de requeriments funcionals a mesura que s'han anat codificant les aplicacions Test.
S'han fet servir dos models de dispositiu *Android*:

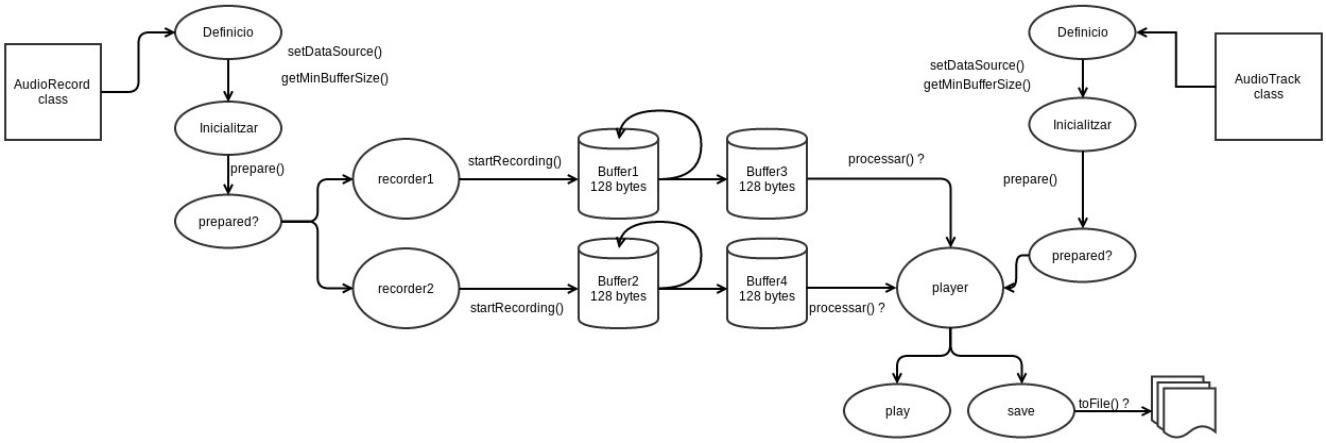
- Smartphone: *Samsung Galaxy Note* II / *HTC Desire*.
- Tablet PC: *XCSOURCE* 10.1”

Per poder explicar millor la evolució en les aplicacions test obtingudes en la fase d'implementació, és presentaran cadascuna d'elles a mode de fitxa d'aplicació resumida amb la essència dels requeriments aconseguits i fins i tot alguna pre-visualització per ajudar a entendre al propòsit i el funcionament de les mateixes. Tot i això s'ha d'afegir que en la codificació d'aquestes aplicacions no s'ha tingut en compte el disseny i s'han utilitzat recursos nadius *d'Android*.

També cal destacar que per aconseguir algunes funcionalitats en les nostres aplicacions s'han desenvolupat aplicatius paral·lels, i es podrien generar algunes aplicacions semblant al prototip que s'ha dissenyat pel projecte (*Breathing* L*ifestyle*). Amb aquesta idea, s'han generat llibreries

3.1 Diagrama inicial:

Des dels inicis de la decisió d'implementar una solució per capturar la respiració en dispositius mòbils, es va dissenyar un model de buffer múltiple, ja que, la càrrega de treball que suposa capturar l'àudio en temps real.
Després d'un procés previ d'estudi de les principals classes *d'Android* que poguessin ser d'utilitat, s'ha fet un diagrama inicial del funcionament general que ha de tenir el sistema de buffer múltiple del sistema de control de respiració.



Com podem comprovar, En el disseny inicial s'han fet servir les classes d'Android:

- AudioRecord
- AudioTrack
- ByteBuffer

La idea principal del disseny del diagrama era processar l'àudio, omplint buffers de 128 *bytes* i de forma paral·lela, clonar cada buffer i reproduir l'àudio instantàniament. Això implica un petit retard, que fa l'efecte d'eco.

En aquesta fase s'han pres diverses decisions, entre elles, la de fer servir Doble *buffer*. El cicle de processament d'àudio pot gestionar-se amb dos buffers treballant en paral·lel, cada buffer serà una instància de la classe *ByteBuffer d'Android*.

Cal destacar que a més de comprendre conceptes més enfocats a la programació, com el doble *buffer*, les classes *d'android* o de *Java*, s'han hagut d'adquirir coneixements sobre conceptes d'àudio com per exemple el '*Samplerate*', freqüència, etc.. En els annexes d'aquest projecte podrem trobar definicions d'aquests així com de les classes Android més utilitzades al projecte.

3.2 Eines de Desenvolupament

A continuació trobem una descripció de les dues eines fonamentals en la programació del projecte, com són, el llenguatge escollit (*Java*) i l'entorn de desenvolupament (*Eclipse*).

3.2.1 Java

Java és un llenguatge orientat a objectes que va aconseguir la seva maduresa amb la popularització d'Internet i que és, en certa manera , l'hereu legítim de *C++* . L'expansió d'aquest llenguatge entre la comunitat de programadors ha estat vertiginosa i s'ha imposat com el paradigma dels llenguatges de programació orientats a objectes . En l'entorn acadèmic i d'investigació , l'ensenyament de *Java* ha substituït (i està reemplaçant) a l'ensenyament de llenguatges de programació estructurada com Pascal i fins i tot *C*, que sempre s'han considerat llenguatges d'elecció per a la introducció a la programació .
De forma resumida , *Java* és un llenguatge neutral , portable, robust , estable , independent de la plataforma , senzill d'aprendre per a programadors que hagin treballat prèviament amb llenguatges orientats a objectes . *Java* pot utilitzar-se per realitzar aplicacions en múltiples plataformes maquinari i sistemes operatius (*Unix* , *Linux* , *OS/390* , *Windows* , o H*P- UX* entre d'altres sistemes operatius per a ordinadors personals o estacions de treball , *Android*, *Palm OS* o *MPOC* entre d'altres sistemes operatius per a dispositius de telefonia mòbil) . Una de les novetats revolucionàries que va introduir Java és la portabilitat . Sun va abordar el problema introduint el model de *bytecode* : quan un programa *Java* es compila no es transforma en un conjunt d'instruccions en codi màquina natives de la plataforma utilitzada , la qual cosa impediria la seva completa portabilitat , sinó que es transforma en un conjunt de *bytecodes* independents de la plataforma utilitzada que són llegits i interpretats per la màquina virtual *Java* , *JVM* , per executar el programa . Per exemple , quan es compila un programa Java en una plataforma *Windows* / *Intel* , s'obté la mateixa sortida compilada , el mateix *bytecode* , que en un sistema *Macintosh* o *Unix*.
Els requisits de desenvolupament per *Android* exigeixen l'ús del *JDK* (*Java Development Kit*) en la seva versió 6 .
El desenvolupament per a aplicacions *Android* es realitza de forma comuna en Java , encara que existeix la possibilitat de realitzar part d'una aplicació mitjançant altres llenguatges mitjançant el set d'eines *NDK* (*Native Development Kit*) encara que no és recomanable pel fet que augmenta la complexitat del desenvolupament de les aplicacions . Seguint les recomanacions de *Google* el projecte farà servir el llenguatge de desenvolupament *Java* .

3.2.2 Eclipse

Eclipse és una plataforma de desenvolupament de codi obert basada en Java. Eclipse va ser desenvolupat originalment per *IBM* com el successor de la seva família d'eines per *VisualAge*. Actualment és desenvolupat per *l'Eclipse Foundation*, una organització independent sense ànim de lucre que fomenta una comunitat de codi obert.
En si mateix *Eclipse* és un marc i un conjunt de serveis per construir un entorn de desenvolupament a partir de components connectats, plugins ... Hi ha plugins per al

desenvolupament en *Java*, *JDT Java Development Tools*, així com per al desenvolupament en altres llenguatges com *C / C + +*, *PHP*, *Cobol*, plataformes com *Android*, etc.
Eclipse és el *IDE* recomanat per al desenvolupament sobre *Android* i s'inclou de facto en el paquet *Bundle* de les *Android Developer Tools*, de manera que el projecte farà ús d'aquesta eina es complementa amb els plugins necessaris per facilitar el desenvolupament del mateix.

3.2.3 Model d'aplicació TEST:

Títol: Aplicació TFC_test1:
Preview: dos o tres fotos de l'aplicació mostrant el fluxe.

Descripció: Aplicació Android que serveis per enregistrar un clip d'àudio en format PCM sense compressió (.wav)

Funcionalitats:
* Accés a disc per emmagatzemar fitxers.
* Enregistrament de clip d'àudio en format .wav
* Reproducció del clip enregistrat

Objectius:
* Aprenentatge d'ús de la classe AudioRecord
* Aprenentatge d'ús de la classe AudioTrack
* Estudi de la estructura de fitxers WAV

Informació Adicional:
* Llibreries adicionals:
* Recursos multimedia:
* Permissos necessaris:
* Nº revisions:

Codi Destacat:
* funció CopyWavetoDisk();
* track.play
* etc...

3.3 Aplicacions TEST

3..3.1 TFC_test1

Títol Aplicació:

TFC_test1

Descripció: Serveix per enregistrar un clip d'àudio en format *PCM* sense compressió (*.wav*). La mida dels *Buffers* es calculen de forma dinàmica en funció de les especificacions d'inicialització de la classe *AudioRecord.* Un cop finalitzem l'enregistrament, es possible reproduir el clip recuperant el fitxer que acabem de guardar al disc.

Preview: Aplicació senzilla amb una única *Activity*, com podem veure els botons que no es fan servir estan inhabilitats:

Funcionalitats:

- Accés a disc per emmagatzemar fitxers.
- Enregistrament de clip d'àudio en format *.wav*
- Reproducció del clip enregistrat

Objectius:

- Aprenentatge d'ús de la classe *AudioRecord*
- Aprenentatge d'ús de la classe *AudioTrack*
- Estudi de la estructura de fitxers *WAV*

Informació Addicional:

- Llibreries addicionals: No
- Recursos multimèdia: Nadius
- Permisos necessaris:
  android.permission.WRITE_EXTERNAL_STORAGE
  android.permission.MODIFY_AUDIO_SETTINGS
  android.permission.RECORD_AUDIO

Codi Font Destacat:

MainActivity.java

```
private static final int RECORDER_BPP = 16;
private static final String AUDIO_RECORDER_FILE_EXT_WAV = ".wav";
private static final String AUDIO_RECORDER_FOLDER = "tfc";
private static final String AUDIO_RECORDER_TEMP_FILE = "record_temp.raw";
private static final int RECORDER_SAMPLERATE = 44100;
private static final int RECORDER_CHANNELS = AudioFormat.CHANNEL_IN_STEREO;
private static final int RECORDER_AUDIO_ENCODING = AudioFormat.ENCODING_PCM_16BIT;
private static final String TFC= "TFC";
```

– Paràmetres d'inicialització de cal classe *AudioRecord*

MainActivity.java

```
bufferSize = AudioRecord.getMinBufferSize(RECORDER_SAMPLERATE,RECORDER_CHANNELS,RECORDER_AUDIO_ENCODING);
```

– La Funció *getMinBufferSize*() determina el buffer mínim necessari en funció dels paràmetres introduïts.

MainActivity.java

```
private void writeAudioDataToFile(){
    byte data[] = new byte[bufferSize];
    String filename = getTempFilename();
    FileOutputStream os = null;

    try {
        os = new FileOutputStream(filename);
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    int read = 0;

    if(null != os){
        while(isRecording){
            read = recorder.read(data, 0, bufferSize);

            if(AudioRecord.ERROR_INVALID_OPERATION != read){
                try {
                    os.write(data);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        try {
            os.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

– Aquesta funció es clau per entendre l'aplicació, es realitza el següent procediment:

1- S'inicialitza un Buffer de bytes amb la mida calculada amb la funció getMinBuffersize().

2- Mitjançat la funció recorder.read(..) es van llegint els bytes del clip de só i es van enregistrant en el buffer ***data****.*

3- S'intenta emmagatzemar el buffer en Disc, en cas que no sigui possible, el sistema generara una Excepció del tipus IOException.

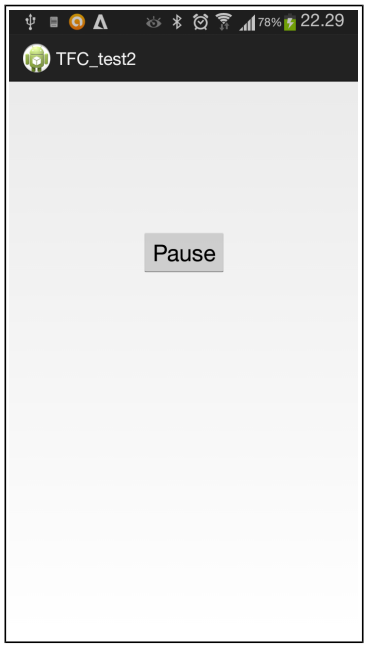
3.3.2 TFC_test2

Títol Aplicació:

TFC_test2

Descripció: Aplicació test per aprendre a treballar en temps real enregistrant àudio i mitjançant les classes *AudioRecord* i *AudioTrack*. El resultat de realitzar el procediment d'enregistrar l'àudio i automàticament reproduir-ho provoca un efecte ECO.

Preview: Aplicació senzilla amb una única *Activity*, com podem veure els botons que no es fan servir estan inhabilitats:



Funcionalitats:

- Lectura de clips d'àudio en memòria (*Buffer* de tipus *byte[]*)
- Reproducció del clip enregistrat automàticament
- Control del procediment mitjançant el botó de *Pause/Play*

Objectius:

- Primera experiència de processament d'àudio en temps real
- Aprenentatge de la classe *AudioTrack*

Informació Addicional:

- Llibreries addicionals: No
- Recursos multimèdia: Bàsics
- Permisos necessaris:
  android.permission.MODIFY_AUDIO_SETTINGS
  android.permission.RECORD_AUDIO

Codi Font Destacat:

MainActivity.java

```
private void recordAndPlay() {
     /* El tipo de dato short es un entero de 16 bits complemento a dos.
     *  Su valor mínimo es -32,768 y el máximo 32,767 (inclusive). Se aplican
     *  las mismas directrices que con byte: puede utilizar short para ahorrar
     *  memoria en grandes arrays, en situaciones en las que el ahorro realmente
     *  importa.
     */
    short[] buffer = new short[1024];
    int num = 0;

    record.startRecording(); //Starts recording from the AudioRecord instance.
    track.play();            //Starts playing an AudioTrack. If track's creation mode is MODE_STATIC,
                             //you must have called write() prior.

    while (true) {
        num = record.read(buffer, 0, 1024);
        //Read 1024 bytes. Reads audio data from the audio hardware for recording into a buffer.
        /* Parameters
        *  -audioData   the array to which the recorded audio data is written.
        *  -offsetInShorts  index in audioData from which the data is written expressed in shorts.
        *  -sizeInShorts    the number of requested shorts.
        */
        track.write(buffer,0, num);
        //Writes the audio data to the audio sink for playback (streaming mode),
        //or copies audio data for later playback (static buffer mode)
    }
}
```

– El procediment *recordAndPlay*() és la part més important del codi font de l'aplicació TFC_test2. Podem identificar clarament el procediment mitjançant el qual el sistema va omplint un buffer de 1024 objectes de tipus *short*.

– S'ha de destacar que el bucle *'While(true)'* està actiu fins que l'aplicació es tanca i per tant es crida a el mètode *Activity.OnDestroy*().

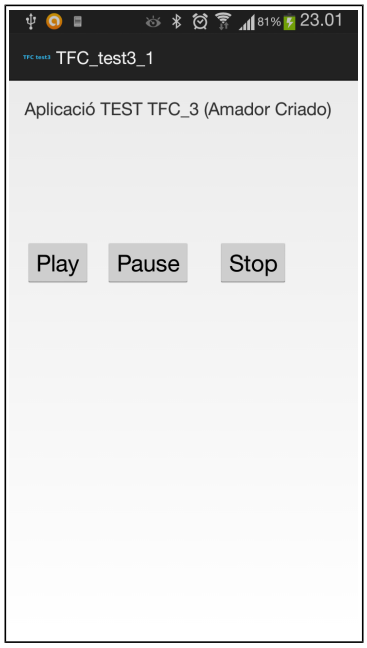
3.3.3 TFC_test3_1

Títol Aplicació:

TFC_test3_1

Descripció: L'aplicació TFC_test3_1 és una evolució de l'aplicació TFC_test3, on s'ha afegit el processat d'àudio en un '*thread*' (fil) dedicat amb la seva pròpia classe**.** El resultat de realitzar el procediment d'enregistrar l'àudio i automàticament reproduir-ho provoca un efecte ECCO.

Preview: Aplicació senzilla amb una única *Activity*.



Funcionalitats:

- Lectura de clips d'àudio en memòria (buffer de tipus byte[])
- Reproducció del clip enregistrat automàticament en 'T*hread*' independent i dedicat
- Control del procediment mitjançant els botons de *Play/Pause/Stop*

Objectius:

- S'ha creat una nova classe dedicada al processament d'àudio i que hereta de la classe nativa de *Java* '*Thread*' .
- Això incrementa el rendiment i augmenta les especificacions d'un disseny orientat a objectes ja que es divideix en classes la implementació de l'aplicació.

Informació Addicional:

- Llibreries addicionals: No
- Recursos multimèdia: Bàsics
- Permisos necessaris:
  android.permission.MODIFY_AUDIO_SETTINGS
  android.permission.RECORD_AUDIO

Codi Font Destacat:

MiHilo.java

TFC_test3_1 ▸ src ▸ com.example.tfc_test3_1 ▸ MiHilo ▸ MiHilo(AudioRecord, AudioTrack, int)

```
package com.example.tfc_test3_1;

import android.content.Context;

public class MiHilo extends Thread{

    AudioRecord record;
    AudioTrack track;
    short buffer[];
    int i;
    long start;
    private boolean continuar = true;
```

– Podem veure com la classe *Mihilo.java* hereta de T*hread* i per tant disposa de tots els mètode si atributs d'aquesta.

MainActivity.java

```
private View.OnClickListener btnClick = new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        switch(v.getId()){
        case R.id.playBtn:{
            log("Start Recording");
            if(isPlaying){
                hilo1.reanudar();
            }else{
                isPlaying=true;
                i++;
                hilo1.start();
            }
            break;
        }
        case R.id.destroyBtn:{
            log("Hilo destruido");
            hilo1.destruir(MainActivity.this);
            break;
        }
        case R.id.pauseBtn:{
            log("Hilo destruido");
            hilo1.parar();
            break;
        }
        }
    }};
```

- Flux principal de funcionament de l'aplicació *MainActivity.java*
- Es defineix un objecte del tipus *View.OnClickListener* per capturar els clics en els botons definits al programa i a partir d'aquí decidir el comportament de l'aplicació.
- Podem comprovar com quan es polsa el botó *'Start'* es llança la execució del objecte de tipus '*thread*' (hilo1) que hem instanciat anteriorment.

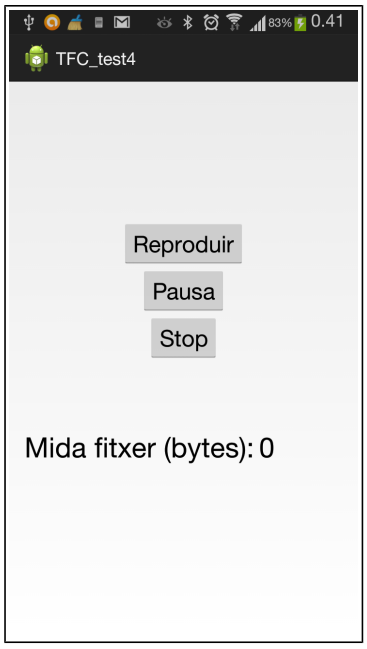
3.3.4 TFC_test4

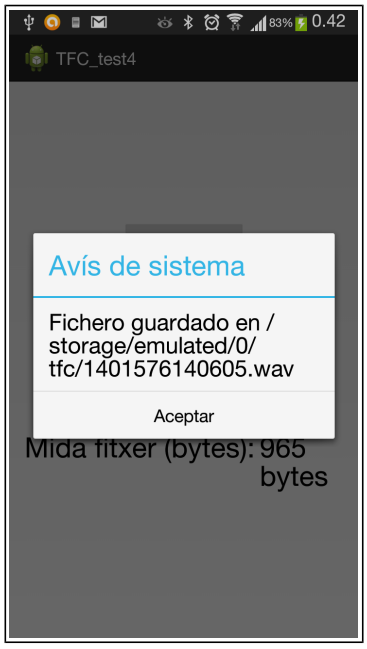
Títol Aplicació:

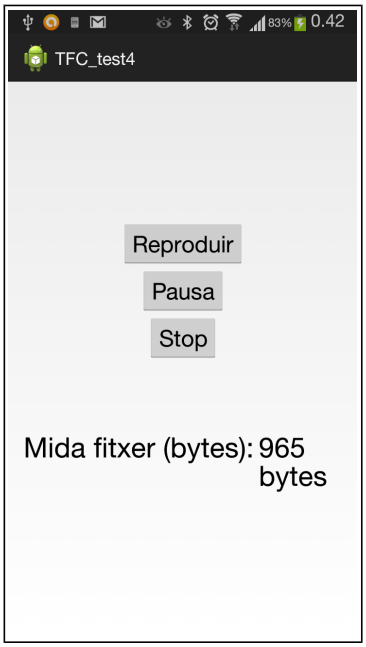
TFC_test4

Descripció: L'aplicació TFC_test4 és una evolució de l'aplicació TFC_test3_1, on per primera vegada s'ha aconseguit un dels objectius més importants del projecte, i no és un altre que el processament mitjançant *doble buffer*. També realitza processat d'àudio en un '*thread*' (fil) dedicat amb la seva pròpia classe**.** El resultat de realitzar el procediment d'enregistrar l'àudio i automàticament reproduir-ho provoca un efecte ECCO.

Preview: Aplicació senzilla amb una única *Activity*. S'ha afegit un
.







Funcionalitats:

- Lectura de clips d'àudio en memòria (buffer de tipus *byte*[])
- Escriptura d'aquests clips d'àudio en disc. L'aplicació a més ens mostra la ruta on

s'han emmagatzemat les dades mitjançant un *Dialog*.
- Reproducció del clip enregistrat automàticament en '*thread*' independent i dedicar
- Control del procediment mitjançant els botons de *Play/Pause/Stop*
- S'ha afegit un *TextView* amb el total de bytes del clip emmagatzemat en disc
- Es realitza tot aquest processament, en temps real i amb doble buffer.
- Es fa servir pe primera vegada l'aplicació *ByteBuffer*, la qual hereta de la classe buffer i podem aprofitar els atributs i mètodes públics d'aquesta classe per el nostre projecte.

Objectius:

- Implementació del primer processament en *doble buffer*
- Aprendre a fer servir la classe *ByteBuffer*.

Informació Addicional:

- Llibreries addicionals: No
- Recursos multimèdia: Bàsics
- Permisos necessaris:
  android.permission.MODIFY_AUDIO_SETTINGS
  android.permission.RECORD_AUDIO
  android.permission.WRITE_EXTERNAL_STORAGE

Codi Font Destacat:

MiHilo.java

```
private void recordAndPlay() throws IOException {


    buffer  = ByteBuffer.allocate(1024);
    buffer2 = ByteBuffer.allocate(1024);
    buffer3 = ByteBuffer.allocate(1024);
    buffer4 = ByteBuffer.allocate(1024);

    String filename = getTempFilename();
    FileOutputStream os = null;
    int num = 0;
    int n=0;
```

– *Buffer* i *Buffer2* són els dos buffers principals, mentre que *Buffer3* i *Buffer4* son buffers auxiliars que serveixen per alliberar el més ràpidament possible als buffers principals i poder processar les dades carregades en memòria en cada iteració.

– En el codi, podem comprovar com els buffers són del tipus ByteBuffer i s'inicialitzen quan es crida a el procediment *recordAndPlay*(). Això es fa d'aquesta forma per mantenir la memòria lliure fins just el moment en que el sistema la necessita.

MiHilo.java

```
while (continuar) {

    if (n % 2 == 0){
        num = record.read(buffer.array(), 0, 1024);

        if(AudioRecord.ERROR_INVALID_OPERATION != num)
        {
            try{
                os.write(buffer.array());
            } catch (IOException e){
                e.printStackTrace();
            }
        }
        BufferTotal.add(buffer.array());
        buffer3 = buffer.duplicate();
        track.write(buffer.array(),0,num);



    } else{
        num = record.read(buffer2.array(), 0, 1024);

        if(AudioRecord.ERROR_INVALID_OPERATION != num)
        {
            try{
                os.write(buffer2.array());
            } catch (IOException e){
                e.printStackTrace();
            }
        }
        BufferTotal.add(buffer2.array());
        buffer4 = buffer2.duplicate();
        track.write(buffer2.array(),0,num);

    }

    track.play();

    n++;

}
```

– Bucle principal de el procediment *recordAndPlay*(), que es crida sempre que es llença l'execució del *thread MiHilo*.

– Cada vegada que es realitza una lectura de 1024 bytes en els atributs buffer i *Buffer2*, automàticament es comprova si hi ha algun error, i sinó es així, s'arxiven en disc i es clonen per el seu processament.

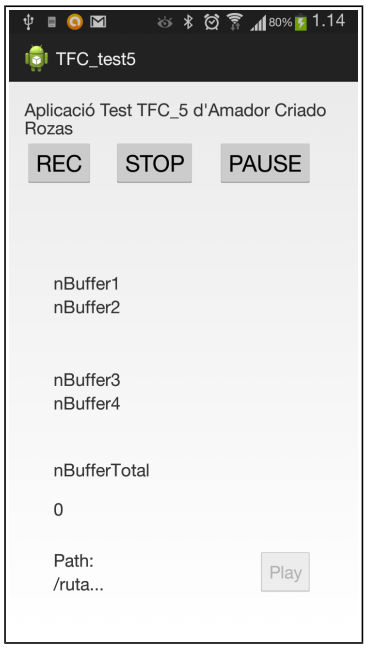
3.3.5 TFC_test5

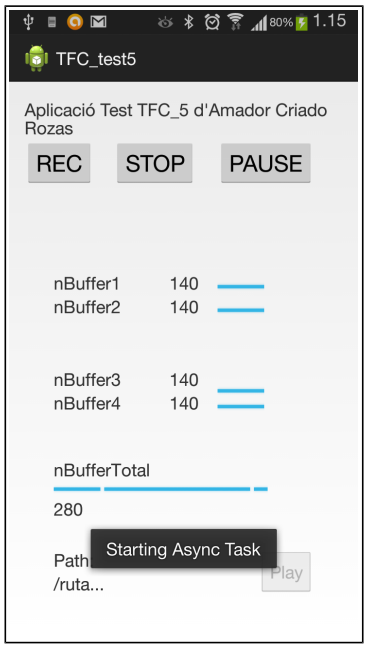
Títol Aplicació:

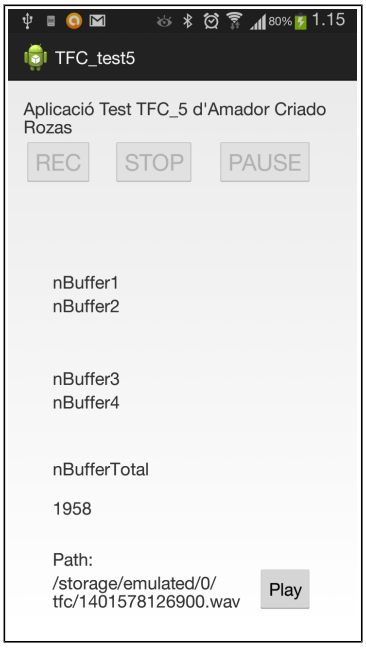
TFC_test5

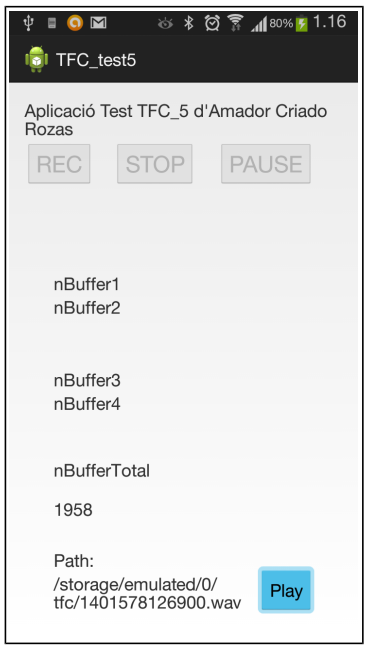
Descripció: De manera simultània es realitzen 3 operacions en temps real. Per una banda, mentre s'està enregistrant l'àudio, es reprodueix i a més, s'actualitzen tots els objectes gràfics que hi han en l'aplicació principal. Podem veure el funcionament del doble buffer més clarament, nBuffer1 correspon amb nBuffer3 i nBuffer2 amb nBuffer4.
Un cop finalitzada la gravació en disc, s'inhabiliten tots els botons excepte del de 'play', de manera que en polsar es reprodueix el clip d'àudio emmagatzemat en disc

Preview: Aplicació senzilla amb una única Activity. S'ha afegit uns elements visuals que mostren el procés d'emplenat dels buffers.









Funcionalitats:

- Lectura de clips d'àudio en memòria (buffer de tipus *byte*[])
- Escriptura d'aquests clips d'àudio en disc. L'aplicació a més ens mostra la ruta on s'han emmagatzemat les dades.
- Reproducció del clip enregistrat automàticament en '*thread*' independent i dedicat. També es pot reproduir recuperant les dades guardades en disc.
- Control del procediment mitjançant els botons de *Rec/Play/Pause/Stop*
- S'ha afegit un *TextView* amb el total de bytes del clip emmagatzemat en disc i un T*extView* per cada buffer, per tant es pot comprovar el funcionament del doble buffer de forma manual.
- S'implementa tota la operativa de doble buffer en un fil dedicat del tipus *AsyncTask*, que es el que comunica el progres de emplenat de cada buffer a la aplicació principal. Un objecte de tipus *AsyncTask* es molt útil per dur a terme operacions atòmiques en una aplicació Android.

Objectius:

- Entendre millor el processament de doble buffer i posar-ho en pràctica, carregant el bucle principal d'operacions per comprovar l'estabilitat del sistema.
- Aprendre a fer servir la classe *AsyncTask*.
- Obtenir una interfície visual més fàcil d'entendre pel que fa al funcionament del doble buffer.
- Alliberar el fil principal d'execució de l'Aplicació (en *Android* Anomenat *UI*) i processar les operacions més costoses en el fil secundari de tipus *AsyncTask*.

Informació Addicional:

- Llibreries addicionals: No
- Recursos multimèdia: Bàsics, s'han inclòs ProgressBar que s'actualitzen en temps real.
- Permisos necessaris:
  android.permission.MODIFY_AUDIO_SETTINGS
  android.permission.RECORD_AUDIO
  android.permission.WRITE_EXTERNAL_STORAGE

Codi Font Destacat:

MainActivity5.java

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main_activity5);
    setVolumeControlStream(AudioManager.MODE_IN_COMMUNICATION);
    setButtonHandlers();
    hilo2 = new MiAsyncTask(MainActivity5.this);

}
```

– Constructor de la *Activity* principal, com podem comprovar, s'inicialitza un objecte (hilo2) de tipus *AsyncTask*, i es passa com a paràmetre la pròpia *Activity* per poder actualitzar les barres de progres dels diversos buffers.

```
private void mostraProgres(){
    p1.setVisibility(View.VISIBLE);
    p2.setVisibility(View.VISIBLE);
    p3.setVisibility(View.VISIBLE);
    p4.setVisibility(View.VISIBLE);
    p5.setVisibility(View.VISIBLE);

    nBuffer1.setVisibility(View.VISIBLE);
    nBuffer2.setVisibility(View.VISIBLE);
    nBuffer3.setVisibility(View.VISIBLE);
    nBuffer4.setVisibility(View.VISIBLE);
}
```

– Exemple de procediment per controlar els elements gràfics de l'aplicació.

```
@Override
protected void onProgressUpdate(Integer[]... values) {
    super.onProgressUpdate(values);

    this.MainTask.getTextView(1).setText(""+values[0][0]);
    this.MainTask.getTextView(2).setText(""+values[0][1]);
    this.MainTask.getTextView(3).setText(""+values[0][2]);
    this.MainTask.getTextView(4).setText(""+values[0][3]);
    this.MainTask.getTextView(5).setText(""+values[0][4]);


}
```

– Les Classes de tipus *AsyncTask* han d'implementar obligatòriament els mètodes, *OnPreExecute*(), *DoInBackground*(), *onProgressUpdate*()..

– En el cas de la porció de Codi afegida, veiem com en cada iteració del bucle principal del fil hilo2, es va augmentant el valor de les *ProgressBar*.

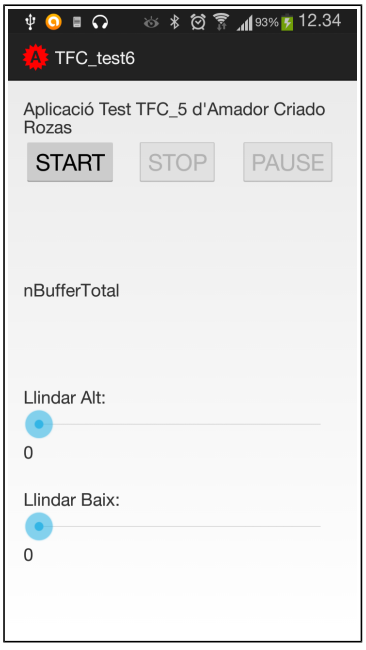
3.3.6 TFC_test6

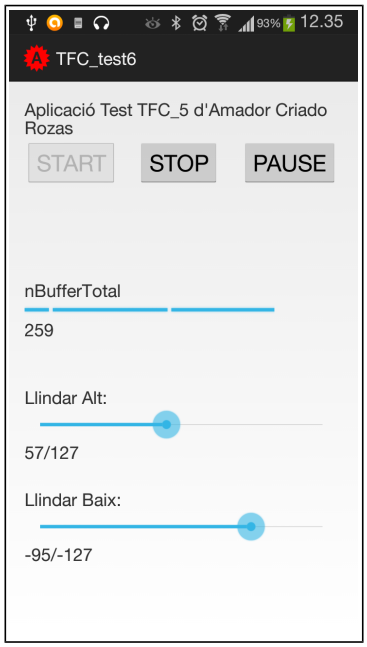
Títol Aplicació:

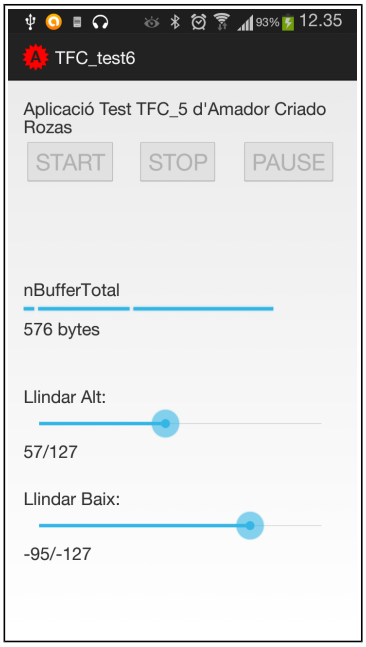
TFC_test6

Descripció: Aquesta Aplicació té com a objectiu principal el processament *byte* a *byte* de la mostra obtinguda en cada iteració. Es a dir, cada vegada que s'omple un buffer, aquest es clona, es processa *byte* a *byte*, i es reprodueix.
Aquesta operativa es possible gràcies al disseny de sistema de doble *buffer*. S'ha introduït dues barres per seleccionar els llindars màxim i mínim. S'ha comprovat que a mesura que els llindars són més propers a 0, es va obtenint una modificació de la veu, semblant a un efecte metàl-lic.

Preview: Aplicació senzilla amb una única *Activity*. S'ha afegit uns elements visuals que mostren el procés d'emplenat dels buffers.





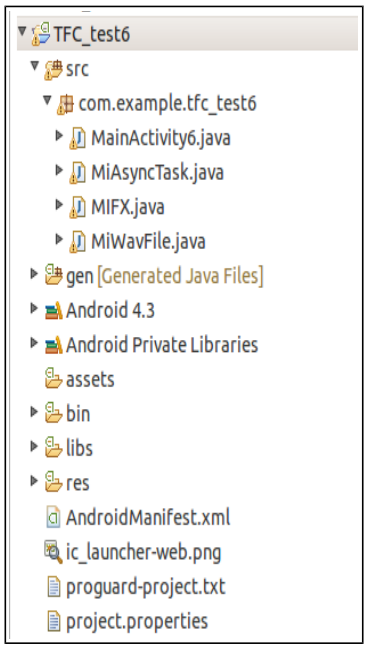


Funcionalitats:

- Lectura de clips d'àudio en memòria (*Buffer* de tipus *byte*[])
- Reproducció del clip enregistrat automàticament en '*thread*' independent i dedicat.
- Control del procediment mitjançant els botons de *Start/Pause/Stop*
- S'ha afegit un *TextView* amb el total de bytes del clip emmagatzemat en memòria, amb la seva corresponent barra de progres..
- S'implementa tota la operativa de doble *Buffer* en un fil dedicat del tipus *AsyncTask*, que es el que comunica el progres de emplenat de cada *buffer* a la aplicació principal. Un objecte de tipus *AsyncTask* es molt útil per dur a terme operacions atòmiques en una aplicació Android.
- Es poden definir el llindar màxim i mínim, es processa aquest llindar en temps real *byte* a *byte*.

Objectius:

- Millorar el processament de *doble buffer* i posar-ho en pràctica.
- Processament *byte* a *byte* de cada *buffer*, un cop s'ha omplert.
- S'ha millorat el disseny orientat a objectes , creant classes específiques per cada funció requerida:



*

screenshot del projecte TFC_test6 a Eclipse

Informació Addicional:

- Llibreries addicionals: No
- Recursos multimèdia: Bàsics, s'han inclòs *ProgressBar* que s'actualitzen en temps

real.

- Permisos necessaris:
  android.permission.MODIFY_AUDIO_SETTINGS
  android.permission.RECORD_AUDIO
  android.permission.WRITE_EXTERNAL_STORAGE

Codi Font Destacat:

MainActivity.java

```
llindarAlt.setOnSeekBarChangeListener(
        new OnSeekBarChangeListener() {

            @Override
            public void onProgressChanged(SeekBar seekBar,
                    int progresValue, boolean fromUser) {
                progressAlt = progresValue;
            }
            @Override
            public void onStartTrackingTouch(SeekBar arg0) {
                // TODO Auto-generated method stub

            }
            @Override
            public void onStopTrackingTouch(SeekBar arg0) {
                // TODO Auto-generated method stub
                hilo2.progress1=progressAlt;
                tLlindarAlt.setText(progressAlt + "/" + llindarAlt.getMax());

            }});
```

– Podem comprovar com, implementant el *listener* del llindar, cada vegada que l'usuari mou la barra de *llindarAlt*, i estableix un valor, s'actualitza l'atribut *progress1* del objecte *hilo1*, i al mateix temps es mostra per pantalla el valor seleccionar amb el mètode *llidaralt*.*setText*().

MiFX.java

```
public ByteBuffer llindarAlt(ByteBuffer b, int limit){

    ByteBuffer b2=b.duplicate();
    byte by;
    int valor;
    byte blimit = (byte) limit;

    for(int i=0;i<1020;i++){
        by=b2.get(i);
        valor=(int) (by);

        if (valor>limit && valor>0){
            b2.put(i,blimit);
        }else {
            b2.put(i,by);
        }
    }
    return b2;
}
```

– La classe *MiFX* ha estat dissenyada per implementar les funcions de tractament de dades o de efectes especials. En aquest cas podem veure la funció llindarAlt, que rep per paràmetre un ByteBuffer '*b'*, el processa *byte* a *byte* i retorna el *ByteBuffer* resultant '*b2*'.

MiAsncTask.java

```
while (continuar) {

    progress1=this.MainTask.progressAlt;
    progress2=this.MainTask.progressBaix*-1;

    if (n % 2 == 0){
        num = record.read(buffer1.array(), 0,2048);
        if(AudioRecord.ERROR_INVALID_OPERATION != num){

            buffer3 = buffer1.duplicate();
            buffer3=fx.llindarAlt(buffer3, progress1).duplicate();
            buffer3=fx.llindarBaix(buffer3, progress2).duplicate();
            BufferTotal.add(buffer3.array());

            if(isPlaying){
                track.write(buffer3.array(),0,num);
                track.play();
            }


        }

        } else{
            num = record.read(buffer2.array(), 0,2048);
            if(AudioRecord.ERROR_INVALID_OPERATION != num){

                buffer4 = buffer2.duplicate();
                buffer4=fx.llindarAlt(buffer4, progress1).duplicate();
                buffer4=fx.llindarBaix(buffer4, progress2).duplicate();
                BufferTotal.add(buffer4.array());

                if(isPlaying){
                    track.write(buffer4.array(),0,num);
                    track.play();
                }
        }

    }

    n++;
    if(isPlaying){
        vBuffer[4]=n;
        publishProgress(vBuffer);
    }
}
```

– Bucle principal de la classe *hilo2*, podem comprovar el procediment de doble *buffer*, i com en cada iteració, es produeixen dos recorreguts de cada *buffer*, ja que s'apliquen les funcions *llilndarAlt* i *llindarBaix*.
– Després d'aplicar el processament, es reprodueix el *buffer* modificat mitjançant la instància *d'ÀudioTrack* '*track*'
– A més, a cada iteració, s'envia informació al *UI Thread*, mitjançant el procediment *publishProgress*(*vBuffer*).
– Es important destacar que el doble buffer s'ha dissenyat dividit en dues iteracions diferents, quan n es parell *(n % 2 ==0)* i quan es imparell *(n % 2 != 0).*
– En aquesta aplicació *test*, la prioritat no es la eficiència del doble *buffer*, sinó posar a prova la capacitat del mateix, per això es fa doble recorregut per cada llindar quan en una versió més eficient es podrien ajustar els dos llindars en un sol recorregut, però no es l'objectiu.

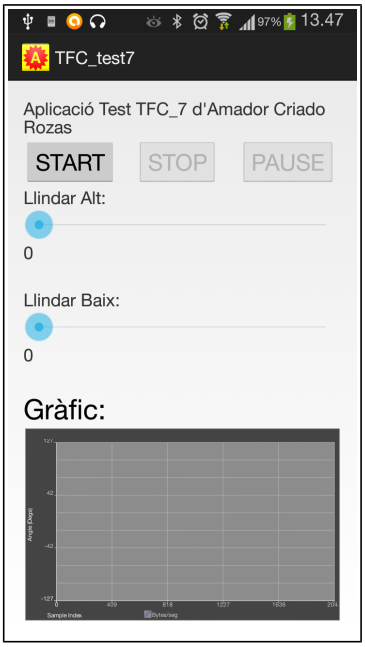
3.3.7 TFC_test7

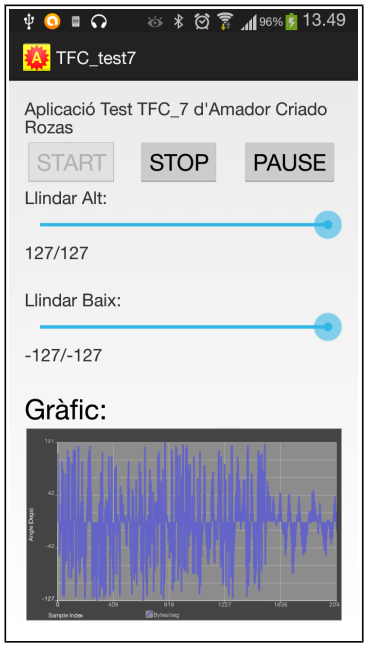
Títol Aplicació:

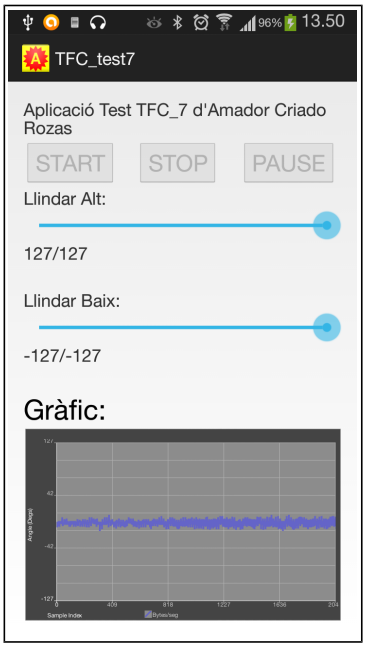
TFC_test7

Descripció: Mostra l'energia del só gràficament.

Preview: Aplicació senzilla amb una única *Activity*. S'ha afegit uns elements visuals que mostren el procés d'emplenat dels buffers i la representació gràfica de la energia del só







Funcionalitats:

- Lectura de clips d'àudio en memòria (*Buffer* de tipus *byte*[])
- Escriptura d'aquests clips d'àudio en memòria.
- Reproducció del clip enregistrat automàticament en '*thread*' independent i dedicat.
- Control del procediment mitjançant els botons de *Start/Pause/Stop*
- S'ha afegit un Gràfic que mostra l'energia del só enregistrat al llarg del temps.
- S'implementa tota la operativa de doble *buffer* en un fil dedicat del tipus *AsyncTask*, que es el que comunica el progres de emplenat de cada buffer a la aplicació principal. Un objecte de tipus *AsyncTask* es molt útil per dur a terme operacions atòmiques en una aplicació *Android*.
- Els *llindars* estan inhabilitats ja que l'objectiu principal d'aquesta aplicació és representar gràficament el só

Objectius:

- Entendre millor el processament de doble *buffer* i posar-ho en pràctica, carregant el bucle principal d'operacions per comprovar l'estabilitat del sistema.
- ,

Informació Addicional:

- Llibreries addicionals: androidplot-core-0.6.0.jar
- Recursos multimèdia: Bàsics, s'han inclòs ProgressBar que s'actualitzen en temps real. També s'han introduït gràfics de les llibreries androidplot.
- Permisos necessaris:
  android.permission.MODIFY_AUDIO_SETTINGS
  android.permission.RECORD_AUDIO
  android.permission.WRITE_EXTERNAL_STORAGE

Codi Font Destacat:

MiAsyncTask.java:

```
protected Void doInBackground(Integer... arg0) {

    record.startRecording();
    this.MainTask.redrawer.start();

    while (continuar) {

        if (n % 2 == 0){
            num = record.read(buffer1.array(), 0,2048);
            if(AudioRecord.ERROR_INVALID_OPERATION != num){
                buffer3=buffer1.duplicate();

                if(isPlaying){
                    ByteDraw(buffer3);
                }

            }

        } else{
            num = record.read(buffer2.array(), 0,2048);
            if(AudioRecord.ERROR_INVALID_OPERATION != num){
                buffer4=buffer2.duplicate();
                if(isPlaying){
                    ByteDraw(buffer4);
                }
            }
        }
        n++;
    }

    return null;
}
```

– En el bucle principal del fil *hilo2* (*AsyncTask*) s'ha afegit en cada iteració una crida a el procediment *ByteDraw*().
– En temps real, cada vegada que es carrega en memòria un *buffer*, aquest es

dibuixa en el gràfic.

MiAsyncTask.java

```
public void ByteDraw(ByteBuffer b){

    int size;
    size=this.MainTask.ByteSeries.size();

    for(int i=0;i<2048;i++){

        if (size > HISTORY_SIZE) {
            this.MainTask.ByteSeries.removeFirst();

        }
        // add the latest history sample:
        this.MainTask.ByteSeries.addLast(null, (int) (b.get(i)));

    }
}
```

– El procediment *ByteDraw*() fa un recorregut *byte* a *byte* del *ByteBuffer* passat per paràmetre, i va actualitzant el gràfic.

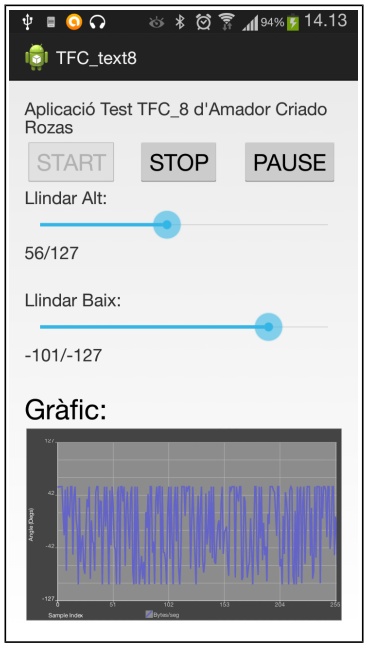
3.3.8 TFC_test8

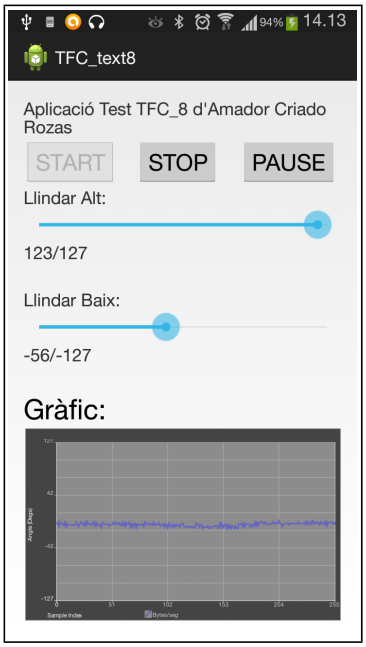
Títol Aplicació:

TFC_test8

Descripció: Podríem definir aquesta aplicació com una evolució clara de la anterior (TFC_test7) però en aquest cas es realitza processament en temps real de les dades carregades als *buffers* mitjançant el procediment de limitació de llindar alt i baix. Només es mostra en el gràfic els valors compresos entre els llindars alt i baix.
L'aplicació ca reproduint el só en temps real amb l'efecte que produeixen els llindars.

Preview: Aplicació senzilla amb una única *Activity*. S'ha afegit uns elements visuals que mostren el procés d'emplenat dels *buffers* i la representació gràfica de la energia del só.





Funcionalitats:

- Les mateixes que l'aplicació *TFC_test7,* l'únic que s'ha afegit és que els llindars estan activats, el que vol dir que en cada iteració del *buffer* es realitza triple processat (*llindaralt*, *llindarBaix*, *ByteDraw*)

Objectius:

- Entendre millor el processament de doble *buffer* i posar-ho en pràctica, carregant el bucle principal d'operacions per comprovar l'estabilitat del sistema.

Informació Addicional:

- Llibreries addicionals: androidplot-core-0.6.0.jar
- Recursos multimèdia: Bàsics, s'han inclòs *ProgressBar* que s'actualitzen en temps real. També s'han introduït gràfics de les llibreries *androidplot*.
- Permisos necessaris:
  android.permission.MODIFY_AUDIO_SETTINGS
  android.permission.RECORD_AUDIO
  android.permission.WRITE_EXTERNAL_STORAGE

Codi Font Destacat:

MiAsyncTask.java

```
protected Void doInBackground(Integer... arg0) {

    record.startRecording();
    this.redrawer.start();

    while (continuar) {


        num = record.read(buffer1.array(), 0,2048);

        if(AudioRecord.ERROR_INVALID_OPERATION != num){
            buffer3=buffer1;
        }

        num = record.read(buffer2.array(), 0,2048);
        buffer3=fx.dobleLlindar(buffer3, progress1, progress2,n).duplicate();

        //** PLAY **//
        track.write(buffer3.array(),0,num);
        track.play();

        if(AudioRecord.ERROR_INVALID_OPERATION != num){
            buffer4=buffer2;
        }


        buffer4=fx.dobleLlindar(buffer4, progress1, progress2,n).duplicate();

        ByteDraw(buffer3,n);

        //** PLAY **/
        track.write(buffer4.array(),0,num);
        track.play();

        ByteDraw(buffer4,n);

    }
    return null;
}
```

- S'ha afegit una funció *doblellindar*() que per cada buffer processa les dades i retorna el *ByteBuffer* passat per paràmetre limitat per els llindars que l'usuari seleccioni.
- Es important destacar que s'han realitzar modificacions en el disseny del doble *buffer* pe millorar la eficiència del sistema. Per exemple s'ha eliminat la comprovació de si *n* era senar o parell, que si que es feia en altres aplicacions Test. Fent aquesta modificació el bucle executa directament les operacions en l'ordre establert i això millor ala velocitat per iteració notablement.

MiFX.java

```
public ByteBuffer dobleLlindar(ByteBuffer b,int limitAlt,int limitBaix){

    int valor;

    for(int i=0;i<2048;i++){
        valor=(int) (b.get(i));

        if (valor<limitBaix && valor<0){
            b.put(i,(byte)limitBaix);

        }else if(valor>limitAlt && valor>0) {
            b.put(i,(byte)limitAlt);

        }
    }

    return b;
}
```

- Processament del doble llindar en un sol recorregut del *BytBuffer* passat per paràmetre. Aquesta es una de les millores més significatives d'aquesta aplicació test, i ha set necessària per mantenir l'eficiència de la mateixa.

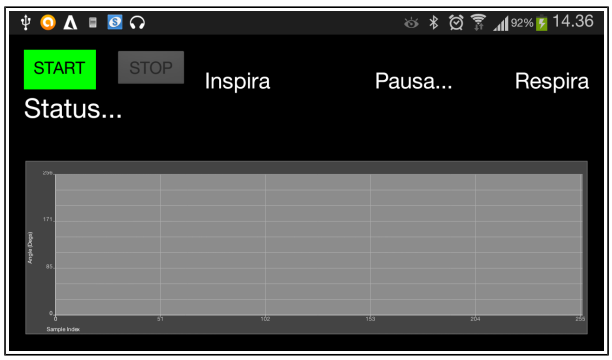
3.3.9 TFC_test9

Títol Aplicació:

TFC_test9

Descripció: L'Aplicació *TFC_test9* és clau en el projecte i és la que comença a tenir més relació amb el prototip dissenyat com a possible aplicació final. Es manté la representació gràfica implementada a les anteriors aplicacions i la gran novetat es que es realitza un calcul d'energia del só en temps real, per tant, l'aplicació sap quan s'esta inspirant, respirant o en pausa.

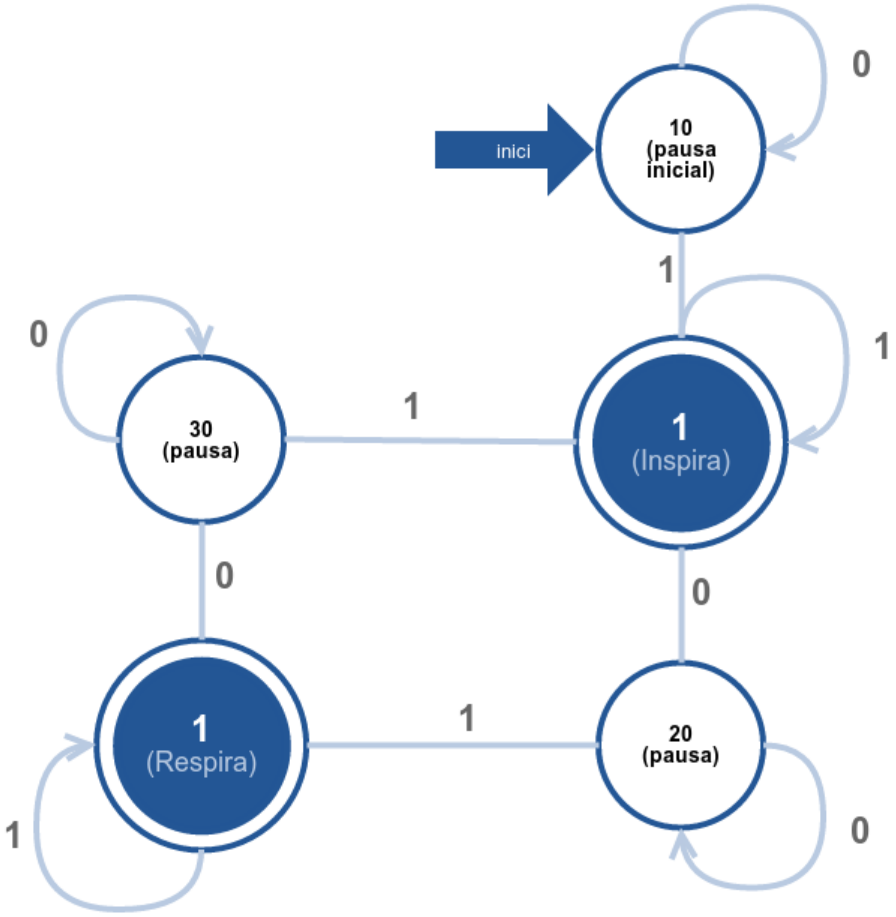
Preview: Aplicació senzilla amb una única *Activity*. S'ha afegit uns elements visuals que mostren el procés d'emplenat dels *buffers* i la representació gràfica de la energia del só.



Funcionalitats:

- S'enregistra l'energia del só en cada iteració, i es calcula la energia mitjana de les últimes 5 mostres.
- Es mostra en el gràfic l'energia del so al llarg del temps.
- Es mostra en quin estat de la respiració es troba l'usuari, mostrant una *progressBar* circular i donant color a el *TextView* de cada estat.
- S'ha dissenyat l'aplicació segons un diagrama d'estats, per controlar els nivells de respiració de manera més eficient:

Diagrama d'estats del control de respiració en l'aplicació TFC_test9:



- Els 1's i 0's signifiquen alta i baixa energia respectivament., cada estat es manté mentes l'energia mitjana de les últimes 5 mostres sigui alta.

Objectius:

- Aplicar els coneixements obtinguts amb el doble *buffer*.
- captar la respiració en funció de l'energia del só.
- Interactuar amb l'usuari mostrant-li en temps real en quin estat de la respiració es troba.

Informació Addicional:

- Llibreries addicionals: androidplot-core-0.6.0.jar
- Recursos multimèdia: Bàsics, s'han inclòs *ProgressBar* que s'actualitzen en temps

real. També s'han introduït gràfics de les llibreries *androidplot*.

- Permisos necessaris:
  android.permission.MODIFY_AUDIO_SETTINGS
  android.permission.RECORD_AUDIO
  android.permission.WRITE_EXTERNAL_STORAGE

Codi Font Destacat:

MiAudioFX.java

```
public int getEnergy(ByteBuffer b){

    int valor;
    ByteBuffer b2=b.duplicate();
    int pos=0;
    int i=0;
    int n=0;


    while(i<2048){
        valor=(int) (b2.get(i));
        if(valor<0){
            valor=valor*(-1);

        }
        pos=pos+valor;
        i=i+48;
        n++;

    }
    pos=(pos/n);
    return pos;

}
```

– En aquesta funció, es passa com a paràmetre un *ByteBuffer* i es retorna un acumulat que ens servirà per calcular la energia del só en funció del seu valor.
– Sabem que el rang d'un *ByteBuffer* es de -127 a 127, tot i això, per a valor negatius, es multiplica per -1 per poder obtenir el valor absolut.

MiAsyncTask.java

```
public void  getMediaEnergy(){

    if(estat==10){
        if(media>80){
            estat=1;
        }
    }
    if(estat==1){
        if(media<30){
            estat=20;
        }
    }
    if(estat==20){
        if(media>80){
            estat=2;
        }
    }
    if(estat==2){
        if(media>30){
            estat=30;
        }
    }
    if(estat==30){
        if(media>80){
            estat=1;
        }
    }

}
```

– Control de flux i de canvi d'estat, en cada iteració

MiAsyncTask.java

```
protected Integer doInBackground(Integer... arg0) {

    audio.startRecording();
    redrawer.start();

    while (continuar) {

        //** Buffer 1 **//
        num=audio.read(buffer1.array(),0,2048);
        buffer3=buffer1.duplicate();

        //** Buffer 2 **//
        num=audio.read(buffer2.array(),0,2048);
        energy=fx.getEnergy(buffer3);


        buffer4=buffer2.duplicate();

        ByteDraw(buffer3);
        ByteDraw(buffer4);

        if (n<5){

        acum=acum+energy;

        }else{

        n=0;
        media=acum;
        acum=0;

        }
```

– Com podem comprovar, a cada iteració es crida a la funció fx.*getEnergy*(*ByteBuffer* b), i es va acumulant l'energia de cada 5 mostres *(if n<5)* per comprovar la mitjana.

MiAsyncTask.java:

```
public void showProgress(int inspira,int pausa,int respira){

    if(inspira==1){
        this.MainTask.ProgressInspira.setVisibility(View.VISIBLE);
        this.MainTask.ProgressRespira.setVisibility(View.INVISIBLE);
        this.MainTask.ProgressPausa.setVisibility(View.INVISIBLE);
        this.MainTask.TextInspira.setTextColor(Color.GREEN);
        this.MainTask.TextRespira.setTextColor(Color.WHITE);
        this.MainTask.TextPausa.setTextColor(Color.WHITE);
    }else if(respira==1){
        this.MainTask.ProgressInspira.setVisibility(View.INVISIBLE);
        this.MainTask.ProgressRespira.setVisibility(View.VISIBLE);
        this.MainTask.ProgressPausa.setVisibility(View.INVISIBLE);
        this.MainTask.TextInspira.setTextColor(Color.WHITE);
        this.MainTask.TextRespira.setTextColor(Color.BLUE);
        this.MainTask.TextPausa.setTextColor(Color.WHITE);
    }else if(pausa==1){
        this.MainTask.ProgressInspira.setVisibility(View.INVISIBLE);
        this.MainTask.ProgressRespira.setVisibility(View.INVISIBLE);
        this.MainTask.ProgressPausa.setVisibility(View.VISIBLE);
        this.MainTask.TextInspira.setTextColor(Color.WHITE);
        this.MainTask.TextRespira.setTextColor(Color.WHITE);
        this.MainTask.TextPausa.setTextColor(Color.RED);
    }

}
```

– El procediment *showProgress*(...) actualitza els elements visuals en temps real.
– Es una funció important ja que es la que fa la feina en *background* per animar l'aplicació e interactuar amb l'usuari final, sense una bona implementació de la mateixa tots els càlculs i dissenys anteriors no tindrien sentit ja que no es captaria en quin estat de la respiració es troba l'usuari.

3.3.10 TFC_test10

Títol Aplicació:

TFC_test10

Descripció: Aplicació en temps real de la transformada ràpida de *Fourier* (*FFT*) a cada *Buffer* carregat en memòria i representació gràfica de l'espectre en el domini del temps.

Preview: Aplicació senzilla amb una única Activity. S'ha afegit uns elements visuals que mostren el procés d'emplenat dels buffers i la representació gràfica de la energia del só.

Funcionalitats:

- Processament del so en temps real, s'aplica la Transformada de *Fourier* de cada *ByteBuffer* i es representa l'espectre en el domini del temps gràficament

Objectius:

- L'aplicació de *FFT* permet analitzar l'àudio obtingut amb més precisió.
- S'ha aconseguit representar l'espectre en el domini del temps per un dels objectius que deriven d'aquesta Activitat es poder representar l'espectre en el domini de la freqüència.

Informació Addicional:

- Llibreries addicionals: androidplot-core-0.6.0.jar, JTransforms-2.4.jar
- Recursos multimèdia: Bàsics, s'han inclòs *ProgressBar* que s'actualitzen en temps real. També s'han introduït gràfics de les llibreries *androidplot*.
- Permisos necessaris:

android.permission.MODIFY_AUDIO_SETTINGS
android.permission.RECORD_AUDIO
android.permission.WRITE_EXTERNAL_STORAGE

Codi Font Destacat:

MiAsyncTask.java

```
while (continuar) {


    num=audio.read(b3,0,2048);

    for (int i = 0; i < 2048 && i < num; i++) {
        b32[i] = (double) b3[i] /127.0 ; // signed 16 bit
    }

    fft.bt(b32);
    num=audio.read(b4,0,2048);

    for (int i = 0; i < 2048 && i < num; i++) {
        b42[i] = (double) b4[i] /127.0 ; // signed 16 bit
    }

    fft.bt(b42);

    ByteDraw(b32);
    ByteDraw(b42);

}
try{
    audio.stop();
}
catch(IllegalStateException e){
    Log.e("Stop failed", e.toString());

}
return 0;
```

– Com podem comprovar en el codi font, es manté el disseny de doble *buffer* però assigne manualment cada valor del *bytebuffer* fent un '*cast*' per convertir cada byte en un objecte de tipus '*int*'
– *fft* es un objecte de tipus *RealDoubleFFT* i mitjançant el procediment f*ft.bt(),* es realitza el '*Backward Tansform*' (Transformació enerera).

4. Resultats

Un cop realitzat l'estudi i disseny previ, els prototips i tota la tasca de programació d'aplicacions test progressives, l'objectiu és poder canalitzar la feina fet cap a una aplicació final o unes llibreries amb les principals funcions utilitzades al projecte. Per aquest motiu, s'ha realitzat un procés de '*Testing*' i de Documentació que tot seguit s'explicaran.

4.1 Avaluació i 'Testing'.

A partir del disseny del prototip inicial, s'han generat un '*sketch*' de l'aplicació '*Bhreathing*

Lifestyle' i s'ha cercat un grup d'usuaris de diferent perfil per provar la funcionalitat de l'aplicació. Tot i que s'ha fet en molt petita escala, recordem que una de les bases del disseny *DCU* és tenir en compte als usuaris en totes les fases de creació del projecte i en la fase que ens trobem, es quan més important seria realitzar aquest '*testing*' a més gran escala i així obtenir resultats més precisos.

Existeixen algunes solucions web per realització de *testing* per part d'usuaris anònims com per exemple *testdroid.com* o *UserTesting.com*, però no s'ha considerat necessari encetar un test a gran escala ja que no s'ha de comercialitzar l'aplicació.

Què és un 'sketch'?

És una simulació de l'aplicació final. L'usuari pot interactuar amb l'aplicació i aquesta compleix amb unes determinades funcions. Es pot desenvolupar en llenguatges com *flash* o *html*. En el nostre cas, s'ha escollit la eina *JustinMind.com* per generar els *sketchs*

4.1.1. Usuaris Test

S'han escollit tres perfils d'usuaris test, el procediment per dur a terme el '*testing*' ha sigut el següent:

- Obtenció del perfil d'usuari
- Breu descripció de l'Aplicació.
- Informar als usuaris de les Tasques a realitzar amb l'Aplicació
- Els usuaris realitzen les tasques
- Cada usuari aporta una resposta o possible *Feed-Back* de cada tasca realitzada

4.1.2. Tasques a realitzar

A continuació detallem les tasques definides per als usuaris en l'aplicació dissenyada mitjançant el 'sketch'. S'han encapsulat en forma de preguntes a l'usuari que realitza el test.

Tasca 1:

Que et sembla el disseny? Et transmet una sensació òptima per concentrar-te o relaxar-te mitjançant la respiració?

Tasca 2:

Completa més de 5 iteracions en mode 'relax'.

Et sents més relaxat?

Quantes vegades ha saltat el l'avís de sistema per no seguir les pautes?

Tasca 3:

Polsa totes les icones de la barra superior.

Et sembla interessant poder compartir l'aplicació amb els teus contactes?

A la pantalla de configuració de l'aplicació, quins elements hi afegiries?

4.1.3. Conclusions 'Testing'

A continuació veurem els resultats obtinguts per els usuaris test

Usuari:	Víctor
Perfil:	30 anys, Enginyer de telecomunicacions Usuari actiu en dispositius mòbils
Dispositiu(s):	Iphone 5 Samsung Galaxy Tab 10.1
Tasca1:	Bé, però per respiració quedaria millor colors blaus. Tot i que el verd es natura.
Tasca2:	Al principi m'ha costat entendre quan havia de inspirar i quan expirar i m'ha saltat l'avís. Un cop solventat aquest problema l'avís no m'ha saltat. He aconseguit relaxar-me, bona experiència.
Tasca3:	Sempre es interessant compartir. A la pantalla de '*Settings*' afegiria la possibilitat de personalitzar l'aplicació, per exemple definir altres colors.
Conclusions:	Tasca1: És interessant i de fet es va valorar fer servir el color blau com a fons. Tasca2: S'hauria de considerar afegir un tutorial quan s'executi l'aplicació per primera vegada. Tasca3: Contemplar possibilitat de poder personalitzar el disseny de l'aplicació a partir de templates pre-dissenyats.

Usuari:	Amelia
Perfil:	34 anys, Administrativa a Cofidis Usuari molt actiu en dispositius mòbils i xarxes socials
Dispositiu(s):	HTC Evo 3D
Tasca1:	Molt bé. M'ha encantat el disseny, és senzill i agradable
Tasca2:	Comencen les pautes de cop, l'aplicació et podria avisar. He fallat en les tres primeres He tingut un problema, quan s'ha separat el telèfon de la meva cara no m'ha registrat bé el só, això seria interessant comprovar-ho. Si, respirar seguint aquesta pauta relaxa, s'ha de reconèixer.
Tasca3:	Si. Cap, ja està bastant complert en la meva opinió.
Conclusions:	Tasca1: Reforça idea de disseny inicial. Tasca2: Afegir un comptador enrere per que l'usuari es prepari abans de respirar. També es podria afegir un control extra mitjançant el sensor de proximitat d'android, de manera que si l'usuari no està davant el telèfon , salta un avís de sistema. Tasca3: Reforça idea de configuració del sistema.

Usuari:	Sergio
Perfil:	23 anys, Tècnic Informàtic Usuari actiu en ús d'smartphones i aficionat als videojocs
Dispositiu(s):	LG optimus II
Tasca1:	Molt bé. Potser posaria les lletres en negre però la resta bé.
Tasca2:	Ho he fet casi perfecte, i crec que pot servir per relaxar-se si està sol i sense distraccions fent servir l'aplicació. M'ha sortit un avís la primera vegada.
Tasca3:	Si. Es podria posar una opció per linkar l'aplicació amb comptes Facebook o Twitter.
Conclusions:	Tasca1: Reforça idea de disseny inicial. Tasca2: el fet de estar sol o no fet servir l'aplicació ens porta ala idea de la evolució que aquesta podria tenir. Gràcies al processat FFT dels buffers podríem detectar soroll de fons i indicar a l'usuari que si necessita relaxar-se hauria de marxar a un entorn més tranquil. Tasca3: Considerar 'linkar' amb xarxes socials, en la pantalla 'splash' s'hauria d'afegir una icona de cada xarxa social (Facebook, twitter,

	google+,...) i poder començar amb un d'aquests perfils, seria una forma de loginar-se. A més és podrien compartir estats en aquestes xarxes socials, com per exemple “Gràcies a l'aplicació Bhreating Lifestyle... Estic relaxat”

Resum de conclusions i possibles millores:

Disseny:
* Valorar fer servir el color blau com a fons.
* Possibilitat de poder personalitzar el disseny de l'aplicació a partir de *templates* pre-dissenyats

Accessibilitat:
* Considerar afegir un tutorial quan s'executi l'aplicació per primera vegada.
* Afegir un comptador enrere per que l'usuari es prepari abans de respirar.
* Afegir un control extra mitjançant el sensor de proximitat d'android, de manera que si l'usuari no està davant el telèfon , salta un avís de sistema.
* processat *FFT*: Podríem detectar soroll de fons i indicar a l'usuari que si necessita relaxar-se hauria de marxar a un entorn més tranquil.

Lifestyle:
* Considerar '*linkar*' amb xarxes socials, en la pantalla '*splash*' s'hauria d'afegir una icona de cada xarxa social (*Facebook*, *twitter*, *google*+,...) i poder començar amb un d'aquests perfils, seria una forma de *loginar-se*
* Podrien compartir estats en aquestes xarxes socials, com per exemple *“Gràcies a l'aplicació Bhreating Lifestyle... Estic relaxa*t”

Com podem comprovar, la potència de la filosofia de disseny escollida és molt potent, ja que ens ha de permetre una evolució molt propera amb l'usuari. Només s'han realitzat tres '*testing*' i s'han estret bastantes conclusions.
Tot i això es interessant considerar altres perfils d'usuari com persones d'avançada edat, persones amb malalties respiratòries, o algun usuari vinculat a l'àmbit mèdic, crec que podrien ser punts de vista interessants.

4.2. Llibreries & Documentació.

Tota la feina d'investigació disseny algorísmic i implementació es pot canalitzar en dues direccions. Per una banda, podríem obtenir l'Aplicació final dissenyada i prototipada i per un altre banda, hem considerat que augmenta el valor de l'estudi realitzat, generar una llibreria estàndard *Java* (**.jar*) i la seva documentació *JavaDoc* corresponent.

S'han definit les següents Classes en la nostra Llibreria:

* DobleBuffer.java
La classe **DobleBuffer** Consisteix en el processat de dos buffers d'àudio en temps real. Instanciant aquesta classe en qualsevol aplicació és pot saber en qualsevol moment si l'usuari està **inspirant**, en **pausa**,o **expirant**.

buffer1: Primer Buffer principal
buffer2: Segon Buffer principal
buffer3Aux: primer Buffer secundari
buffer4Aux: Segon Buffer secundari

* Audio.java
La classe **Audio** és una extensió d'AudioRecord. S'han afegit algunes funcionalitats per adaptar-la a les necessitats del doble buffer i de les característiques del so que emet la respiració.

* FX.java
La classe **FX** conté totes les funcions que processen la informació continguda als buffers

* WavFile.java
La classe WavFile gestiona la lectura/escriptura de fitxers *.wav* en disc.

4.2.1 Comentant amb Javadoc

A continuació podem veure un parell de '*screenshot*' del *JavaDoc* generat de les llibreries.

Index.html

DobleBuffer.html

All Classes
Audio
DobleBuffer
FX
WavFile

Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.example.tfc_tools
Class DobleBuffer

java.lang.Object
└ └com.example.tfc_tools.DobleBuffer

public class **DobleBuffer**
extends

La classe **DobleBuffer** Consisteix en el processat de dos buffers d'audio en temps real. Instànciant aquesta classe en qualsevol aplicació és pot saber en qualsevol moment si l'usuari està **inspirant**, en **pausa**, o **expirant**.
buffer1: Primer Buffer principal
buffer2: Segon Buffer principal
buffer3Aux: primer Buffer secundari
buffer4Aux: Segon Buffer secundari

Since:
02-jun-2014

Version:
1.0

Author:
Amador Criado

Field Summary

Modifier and Type	Field and Description
private int	**acum** The acum.
private Audio	**audio** Instància de la classe **Audio** que hereta d'**AudioRecord**
private int	**audio_encoding** The audio_encoding.
private int	**bpp** The bpp.
private boolean	**breathHold** The breath hold.
private boolean	**breathIn** The breath in.
private boolean	**breathOut** The breath out

4.2.2. Creació d'una llibreria .jar

Tant la llibreria, com la documentació, es torben disponible en el repositori que s'ha creat expressament per aquest projecte:

https://github.com/ameithor/android-breathe-doblebuffer

4.3. Conclusions

Les conclusions que es poden extreure de la elaboració d'aquest projecte són molt positives. Es podria dir que la gran majoria de requisits inicials que es van plantejar quan es va decidir el contingut del projecte s'han assolit i això es pot considerar un èxit.

S'han aplicat tècniques de programació i disseny algorísmic apresses durant el transcurs de la carrera i a més s'han hagut d'ampliar coneixements en altres aspectes més avançats i d'avantguarda tecnològica com és la programació en *Android*.

Particularment, el fet de dur a terme una filosofia de disseny orientada a l'usuari he permès aprendre conceptes com el prototipat o el '*testing*' i s'ha ampliat molt el coneixement sobre dispositius android.
Tornar a realitzar tasques de programació ha significat un retrobament amb conceptes impartits a la carrera, ha de constar que en moltes ocasions ha sigut necessari recuperar alguns apunts i reciclar coneixements.

La Metodologia del mòdul d'implementació, ha servit per desenvolupar diverses aplicacions, i tot i que en el projecte no s'ha donat gaire rellevància a els aspectes visuals de les aplicacions, ha sigut una part que s'ha hagut de considerar i evidentment s'han dissenyat les aplicacions de manera que els controls de flux siguin útils i això permetés un aprenentatge àgil.

Un cop acabat el projecte, podríem dir que el principal objectiu ha sigut el de desenvolupament a partir del disseny, la investigació, i la interacció directa amb l'usuari fet que es important si s'enfoca l'estudi de qualsevol carrera a el món laboral, on l'usuari (o en el cas d'una empresa, el client) té una importància molt elevada, fins i tot és el nucli per moltes organitzacions.

La confecció d'una documentació ampliada i unes llibreries amb format estàndard per poder ser importades en qualsevol projecte, determinen la intenció de que el projecte tingui un caire de investigació i recerca de recursos. En aquest sentit, també he après molt sobre programació col-laborativa i documentació.
Es possible introduir diverses millores en el projecte, de fet, a l'apartat de 'testing' hem pogut comprovar com amb només tres tests a usuaris de diferent perfil s'han obtingut diverses propostes de millora. Tot i això, està clar que una de les millores més importants seria poder captar la respiració en funció de la representació gràfica de la freqüència de cada buffer, ja que mitjançant *FFT* hem aconseguit obtenir arrays de nombres complexos e inclús representar-los al gràfic, però realitzar aquest processament es de una dificultat elevada i s'escapa de la cobertura d'aquest projecte.

Un altre objectiu que deriva del projecte és el de crear l'Aplicació dissenyada, i publicar-la al *market d'Android*, i que definitivament, es faci realitat tot el disseny i implementació prèvia, amb la importància que tindria per mi que la feina realitzada pogués arribar a usuaris de qualsevol part del planeta.

4.4. Agraïments

És difícil agrair una feina que s'ha dut a terme durant anys i no deixar-se a ningú, però després de començar a treballar abans de finalitzar la carrera i quedar pendents algunes assignatures i el projecte que he hagut de fer a distància, haig de dir que ha sigut molt complicat i que compaginar feina i estudis s'ha fet molt més dur del que em pensava la principi.

A la meva mare **Paqui**, gràcies per haver-me donat la millor educació possible i recolzar-me sempre.
Al meu germà **Sergi**, mil gràcies per fer-me sentir admirat i recolzat, espero que et vagi molt bé.
A la meva xicota **Amelia**, sense ella, no hagués estat possible aconseguir finalitzar la carrera, m'ha recolzat en tot moment i ha set molt comprensiva amb la meva situació. En aquest treball hi ha una part important d'ella.
Al meu gat **Yomsi**, que ha escalfat el meu seient de feina mentre jo no estava.
A tots els **companys de feina** que he tingut tots aquests anys, he après molt de tots ells i aquest aprenentatge està plasmat en el projecte.
A tots els **companys i professors** de la universitat.

A tots ells.. GRÀCIES,

Agraïment especial a una persona que ha sigut un model i un exemple per mi i que m'ha animat durant mols anys a no deixar de banda la carrera i a finalitzar-la. Sense els seus consells i la seva exigència no hagués continuat la carrera i segurament hagués desistit d'estudiar i m'hagués centrat en treballar... ***al meu pare Amador****, GRÀCIES, t'estimo i mai t'oblidaré.*

5. Bibliografia

Manual Imprescindible JAVA 7
F. Javier Moldes Teo

Web oficial de Desenvolupadors Android
http://developer.android.com/sdk/index.html

Tècniques de respiració
http://www.psicologia-online.com/autoayuda/relaxs/respiracion.htm

Format fitxer WAV
https://ccrma.stanford.edu/courses/422/projects/WaveFormat/

StackOverFlow
http://stackoverflow.com/
S'han obtingut molts recursos d'aquesta web, es tracta d'un fòrum orientat a preguntes/respostes de programació i hi ha moltíssim contingut d'android i Java.

Java optimization rules.
http://www.appperfect.com/
Web que s'ha fet servir per augmentar la eficiència del codi font.

Blog de SGOliver
http://www.sgoliver.net/
Blog molt interessant amb molts exemples pràctics de desenvolupament Android.

JustInMind
http://www.justinmind.com/
Imprescindible els seus tutorials per aprendre a realitzar prototips.

Fisioteràpia respiratòria
http://www.fisioterapiarespiratoria.es/

6. Annexes

6.1. Conceptes Imprescindibles

Farem un repàs a les principals funcions I classes d'Android que s'han fet servir. D'aquesta forma serà més fàcil entendre les diferents funcionalitats de les aplicacions de 'testing' i de les llibreries resultants

6.1.1 AudioManager

AudioManager proporciona accés a control de volum i de timbre. És important tenir accés a aquesta classe ja que serà la que ens donarà accés al elements tan importants com el micròfon.

6.1.2. A*udioRecord*

La classe AudioRecord gestiona els recursos d'àudio d'aplicacions Java per realitzar gravacions d'àudio des de el corresponent hardware d'entrada, com per exemple micròfons integrats o externs. Aix+o s'aconsegueix mitjançant “pulling” (lectura) de dades d'un tipus d'objecte Audiorecord. L'aplicació és la responsable de realitzar el “pulling” a temps del tipus d'objecte audiorecord fent servir un dels següents tres mètodes:

1- read(byte[], int, int)

2- read(short[], int, int)

3- read(Bytebuffer, int)

La elecció de quin dels tres mètodes anteriors s'ha de fer servir es basarà en el format de l'àudio que emmagatzemarem ja que és el mes convenient per d'utilització d'audiorecord

En el moment de creació d'un objecte audioRecord, aquest inicialitza el seu àudio buffer associat que s'omplirà amb les noves dades d'àudio. La mida d'aquest buffer, s'especifica durant la construcció, determina la mida màxima que un objecte de tipus Audiorecord pot enregistrar abans de produir un “over-running” de dades que encara no han estat llegides. Les dades han de ser llegides de l'àudio Hardware en paquets de mida inferior a la del buffer.

6.1.3 AudioTrack

La classe Audiotrack s'encarrega de gestionar i de reproduir recursos d'àudio per aplicacions Java. Audiotrack permet “streaming” de PCM àudio buffers cap a els destinataris (altaveus) d'àudio per reproduir-ho. Això s'aconsegueix mitjançant “pushing” la informació al objecte de tipus AudioTrack fent servir un dels següents mètodes:

1- write(byte[], int, int)

2- write(short[], int, int)

Una instància d'AudioTrack por operar amb dos models diferents, estàtic o “streaming”. En el mode “streaming” l'aplicació escriu un flux continu de dades cap al objecte tipus Audiotrack, fent servir un dels existents mètodes 'write()'. Aquests, bloquegen la informació fins que és transmet des de la capa Java fins a la capa nativa per encauar la reproducció.
El mode 'streaming' és més útil quan es reprodueixen blocs d'àudio que per exemple són:

– Massa gran per encaixar en memòria degut a la duració del só per reproduir-se.
– Massa gran per encaixar en memòria degut a les característiques de l'àudio (alt 'àudio sampling rate, bits per mostra,...)
– Rebut o generat mentre prèviament l'àudio encuat s'està reproduint.

El mode estàtic hauria de ser escollit quan es tracti amb sons de curta durada que encaixen en memòria I que necessiten ser reproduïts amb la més petita latència possible. El mode estàtic serà a més a més, el més adequat per UI(User interface) I sons de jocs que han de ser reproduïts molt sovint, I amb la menor sobrecàrrega possible.

Un cop creat, un objecte audioTrack inicialitza el seu buffer associat. La mida d'aquest 'buffer', especificada durant la construcció, determina la durada que AudioTrack pot reproduir abans de quedar-se sense dades.
Per un Audiotrack fent servir el mode estàtic, la mida del 'buffer' és la mida màxima del só que pot ser reproduïda .
En el mode 'streaming', les dades seran escrites al destinatari d'àudio en porcions de mides de menys o igual que el total de la mida del buffer. AudioTrack no és definitiva i por lo tant permet subclasses, però el seu ús no és recomanat.

6.1.4 ByteBuffer

Aquesta classe deriva directament de la classe **java.nio.Buffer** de manera que hereta totes les seves propietats però a més es tracta d'un cas de Buffer especific, els components del gual son objectes de tipus byte.
Es important assignar memòria quan creem un objecte de tipus ByteBuffer. Podem fer-ho, o bé reservant l'espai necessari mitjançant el mètode **allocate(n)** on n fa referència als n bytes que necessitarem, o si en el nostre codi disposem d'un array de bytes del tipus byte[], podríem fer servir la funció wrap(byte[]) per assignar tot el array de byte a l'objecte byteBuffer. Aquesta segona opció és molt més interessant a nivell de gestió de memòria

en temps d'execució ja que mantenim la memòria que necessitarà el ByteBuffer fins a que realment ho necessitem.

Aquesta classe ens ha permès fer servir conjunt de dades d'una forma àgil I ens ha estalviat algunes implementacions que haguessin sigut necessàries, com per exemple, la inserció directa d'algun element en la posició i-éssima del Buffer.

6.1.5 AndroidPlot

AndroidPlot és un API per a la creació de gràfics dinàmics i estàtics dins d'aplicacions Android. Va ser dissenyat des de zero per a la plataforma Android, és compatible amb totes les versions d'Android des 1.6 d'ara endavant i és utilitzat per més de 500 aplicacions en el mercat actual.
S'ha escollit aquesta llibreria externa perquè s'ha decidit invertir més recursos de desenvolupament en altres aspectes com per exemple aconseguir el processament de les dades en temps real mitjançant el doble Buffer.
Ha sigut de gran utilitat per interpretar el processament obtingut sobre les dades, sobretot quan s'ha aplicat FFT. S'han fet servir XYSeries i gràfics dinàmics tant en domini del temps com de freqüència.

6.1.5 Jtransforms

JTransforms és el primer biblioteca FFT multiprocés de codi obert escrit en Java pur. Actualment, quatre tipus de transformades estan disponibles:
transformada discreta de Fourier (DFT),
Transformada Discreta del Cosinus (DCT),
transformada sinusoïdal discreta (DST)
discrets Hartley Transform (DHT).
El codi es deriva de l'ús general FFT paquet escrit per Takuya Ooura i des de Java FFTPack escrit per Baoshe Zhang.

6.2. Conceptes Bàsics digitals de so

Per entendre i poder avançar en el desenvolupament del projecte, s'han estudiat conceptes bàsics relacionats amb el só digital, a continuació es definiran:

6.2.1 Freqüència

És el nombre de vibracions porció segon que dóna origen al so analògic . L'espectre del so es caracteritza pel seu rang de freqüències . Aquesta es mesura en hertz (hz) . L'oïda capta només aquells sons compresos en el rang de freqüències entre 20 hz i 20.000 hz

6.2.5 cbr / vbr

Velocitat de bits constant / variable . ***Cbr*** indica que l'àudio ha estat codificat mantenint el bitrate constant al llarg del clip d'àudio de mentre que vbr varia entre un rang màxim i mínim en funció de la taxa de transferència .

6.2.6 còdec

Acrònim de " codificació / descodificació " . Un còdec és un algorisme especial que redueixen el número de bytes que ocupa un arxiu d'àudio . Els arxius codificats amb un còdec específic requereix el mateix còdec per ser i descodificats reproduïts i . El còdec més utilitzat en àudio és l'mp3 .

6.2.8 PCM

Mètode que es fa servir per representar digitalment mostres de senyals analògiques. En un flux de PCM l'amplitud de la senyal analògica es mostrejada regularment en intervals uniformes, on cada mostra es quantificada al valor més proper del rang de valors digitals. Linear PCM és un tipus especific de PCM en el que la quantificació de nivells és linealment uniforme.

Els fluxos PCM tenen dues propietats bàsiques que determinen la seva fidelitat envers de la senyal analògica original.

– Sampling rate: Nombre de vegades per segon que es prenen les mostres.
– Bit depth: Determina el possible nombre de valors digitals que cada mostra pot prendre.